UNIVERSITY OF OSLO
Department of Informatics

Analysing the Scalability of Virtualized Environment

Master Thesis

Bishmaya Banstola

Network and System Administration

May 23, 2012

Analysing the Scalability of Virtualized Environment

Bishmaya Banstola

Network and System Administration

May 23, 2012

Abstract

The necessity of allowing time-sharing a single computer between multiple, single tasking operating systems emerged the concept of virtualization. As a result, old concept of one-operating system for one-server paradigm is run-down. Of course, proper division of resources in a single machine for multiple operating systems is a tremendous job of virtualization, and hence it is flourishing among computer users. On the other hand, it is making the users confused to select the best from the pool of good operating systems, hypervisors and other tools and techniques. Similarly, choosing the number of virtual machine is not only a complex issue for them, but also the performance of the whole environment may go down on wrong judgement. Rather than doing scientific analysis on these matters, usually decisions are made on general discussion and perception.

Therefore, it is immensely desired to understand the performance impact on a virtual machine on the presence of its neighbours. This thesis scientifically evaluates and figures out the scalability impact on a virtual machine. The chosen operating system is one of the most popular operating systems, i.e., Debian GNU/Linux OS. Similarly, KVM (Kernel-based Virtual Machine) is chosen as a virtual machine monitor. The performance of the virtual machine is analysed by running series of benchmarks one by one. First the system is measured as a white box for analysing its micro-performance, and then it is measured as a black box for macro-performance. The finding of this research not only delivers an ideal pattern of system performance, but also gives a clear vision to the system administrators and/or organizations to understand how the scalability of machines impacts performance of the system, which component is less affected and which is affected more.

Acknowledgements

I would like to express my sincere gratitude and respect to my advisor Mr. Ismail Hassan for his unfailing support, granted excuses, unlimited patience and continuous encouragement throughout the research period. I am also thankful to Mr. Martin Kot and Mr. Amir Maqbool Ahmed for their help for troubleshooting the network and firewall problem in HioA during my work.

I also want to express heartfelt appreciation to my friends Anunaya Banstola and Binaya Banstola for their thoughtful cooperation during the experiments. I am grateful to Ashru, Reeya, Kusum and other friends for their all help and support.

Finally, I would like to express my deep gratitude to all my family members whose constant support and eternal blessings are the keys for all the success in my life.

May 23, 2012
Bishmaya Banstola

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Virtualization techniques started to accelerate very rapidly after its revival. Within a few years, large numbers of hypervisors[1] are developed. Usually development is focused on virtual machines monitors only but there are verities of interest of virtual machine users which are based on their own requirements, and they do not want performance degradation [1] on their virtual machines. A system administrator task is not only finding cost effective methods and tools, but also the responsibility to choose tools that gives optimal performance. Rather than choosing haphazard approach, a system administrator has to analyze some major challenge him/herself, such as Is the performance of virtual machine affected on the presence of other machine(s)? How does the performance of the system fluctuate? What factors lead to the performance loss? Does the behaviour of virtual machine depend on the operating system which is running into it? Or does virtualization deployment technique affect the performance of Virtual Machine? There are many yes/no answers. As we know, virtualization is a smart concept of allowing multiple instances of operating systems to run simultaneously on single physical machine but none of us think about how many virtual machines are suitable to run on the virtualization tool of interest. When several virtual machines are concurrently running on the same physical host, each virtual machine may exhibit an unstable performance [7] [8]. This is because of the workload imposed on the system by

[1] *Hypervisor is a Virtual Machine Monitor which creates virtual environment to run multiple operating systems to share a same hardware resource*

other VMs[2], virtualization technique, installed operating system and so on. We are usually careless about these things, and hence we forget major issues.

The main motivation of this research rounds on the issues described above and also there is a lack of appropriate research for the comparative analysis of virtual machines performance on scalability. Without having a clear vision on these matters, system administrators will not be able to deploy correct tools for optimistic result. More than this, performance of whole virtual environment may get ruin, if the number of virtual machines is mistaken. Thus, a scientific investigation for the performance difference due to scalability of virtual machine is ultimately needed. As the result of this thesis gives a pattern of system fluctuation, it would be a remarkable reference to researchers who want to investigate other virtual environments composed of different hardware and software components. This issue intensively inspired to do this research.

1.2 Problem Statement

The core problem statement of this research is as follows:

How does the performance of virtual machine degrade on the presence of other virtual machines under KVM (Kernel-based Virtual Machine) environment?

Performance of the system is said to be best if the system has better resource utilization, balanced CPU usages, high throughput and needs less time to solve a given task even in the existence of large number of virtual machines.

Virtualization tools facilitate to create virtual machines. **Virtual machine** is not a real machine. It is simulated by software. Virtual machines do not know the presence of other virtual machines. They are completely isolated to each others.

Virtual Environment is a such environment on which virtualization technique is implemented to manage the large number of virtual machines, and the condition in which several virtual machines run side by side on the same virtualization platform is referred as the **scalability** of virtual environment.

[2] *VMs: Virtual Machines are virtual environments created by Virtual Machine Monitor (VMM)*

Degradation of virtual machine performance can be defined as the declination to the sate on which the system is performing low. It means, virtual machine is not giving its output in that way on which it can deliver optimum performance.

KVM is a latest generation of open source hardware based full virtulaization solution. It supports execution of multiple virtual machines with unmodified guest operating systems, like Linux and Windows.

1.3 Assumption and Limitation

The main objective of this research is to find out the best and reasonable result for the problem stated above, and this is done by the scientific calculation of extracted data throughout the experiment. The result is neither twisted nor influenced by anything. This research ensures fair analysis.

Throughout the whole experiment, each phase of the tests are done on the same hardware configuration. So that all virtual machines always share same the hardware resource, which creates a fundamental background to evaluate the performance and the behaviour of virtual machines running on it. It is also supposed that all virtual machines are identical to each other with the same system configuration. The systems do not provide any service, like web, database etc. Only minimal packages are installed in the system to avoid unnecessary system resource consumption. To ensure the optimal outcome, entire virtual machines are flushed and recreated before setting-up new phase of benchmarking.

Even-though this experiment is done among the large number of virtual machines, the guest operating system of each virtual machines would be Debian GNU/ Linux. Similarly, Kernel Based Virtual Machine (KVM) would be only a virtual environment for this research. These two are the main limitations of this research. The output of this research is gained from the pre-considered environment (Debian guest machines on KVM), so it may not have very meaningful to another virtual environments. It is because, different underlying hardware, guest operating system and number of virtual machines give dif-

ferent result, but it would give a great reference to researchers to have a clear vision before starting their research on different environments. The conclusion of this experiment is presented on the basis of macro-benchmarking and micro-benchmarking results (section 1.2).

1.4 Hot Topics and Related Works

After one-operating system for one-server paradigm, a new era has begun with virtualization. Hence, interests of researchers toward the performance of virtual machines are risen very high. Varieties of hypervisors and virtualization applications are promoting them for the performance analysis of computer system.

Generally most of researches are conducted for the performance analyzing of one particular tool (example: [16] [20]). However, several high-level and low-level performance analyses are carried out heavily from the last few years [10] [11] [12] [17] [19].

Andrew Whitaker et. al. have described about a layer of indirection on their research paper '*Rethinking the Design of Virtual Machine Monitors*' [3]. This layer adds ease to inspect virtual machine performance easily by realizing intrusion detection [5], performance analysis [36], live migration [2] [4], fault tolerance etc.

'A comparative study of Virtualization of Linux Servers' [28] by Girard et. al. have analyzed six virtualization technologies like KVM, VirtualBox, Xen and other. The research is focused on scalability of respective virtualization tools. The performance analysis is carried out with the help of micro-benchmarking work load.

In the paper *'Discovering hypervisor overloads using micro and macro benchmark'* by Andrea et. al. [26] have run series of macrobenchmark and macrobenchmark to analyze the performance of KVM, Xen and and Virtual Box in hosted virtualization architecture.

Moses Mungai in his thesis in University of Oslo entitled '*Performance Analy-*

sis of Different POSIX[3] *Operating Systems as Virtual Machines*'[12] has conducted macro and micro benchmarking phase to determine the best Operating System that is best suited for certain roles like network intensive services, file system and I/O intensive services.

Jianhua Che et. al. [6] in *'A Synthetical Performance Evaluation of OpenVZ, Xen and KVM'* have mentioned their clear procedure and result of benchmarking to analyse performance of openVZ, Xen and KVM hyprevisors. In their research, a combine result of processor performance, Network performance, Database server performace, disk performance have reflected the overall system performance. Low-level benchmarking like context switching has revealed the micro performance of system.

Dawei Huang et. al. [9] have described benchmarking tools; Linpack, Lmbench and IOzone in their paper entailed '*Performance Measuring and Comparing of Virtual Machine Monitors*'. They have provided a series of performance experiment during the testing of Xen and KVM hyprevisors. By doing CPU overhead analysis, memory bandwidth analysis, I/O operating analysis, they have tried to figure out main source of the total virtualization overhead.

Margo I. Seltzer et. al. team in their paper '*A Case Study of the Performance of NetBSD on the Intel x86 Architecture*' [13] have explained about lmbench and hbench for detailed analysis and decomposition of the performance of operating system primitives. They have advice to centralize on operating system performance rather than concentrating on application-level performance. As we have such tools to reveal the architectural dependence of OS performance.

J. Bradley Chen et. al. team in their paper '*The Measured Performance of Personal Computer Operating Systems*' [15] have proved that the performance of operating systems is affected by number of structural issues, such as system hook call, machine mode change, memory segmentation. They have used microbenchmark to show system difference and application workloads to expose end-to-end performance.

[3] *POSIX (Portable Operating System Interface) is a standard specified by the IEEE (Institute of Electrical and Electronics Engineers) for maintaining compatibility between operating systems*

In the paper '*Xen and the Art of Virtualization*' [14] by Paul Barham et. al. have done comparative analysis of Xen performance where they determined the excellent performance of Xen on network-centric services, such as, local mirroring [25] of dynamic web content, media stream transcoding [21] and distribution, multi-player game [22] and virtual reality servers.

Still, none of the researchers are aware to address some major issues of virtualization, such as consolidation issue, benchmark issue, interference issue etc.

- Consolidation:

  Is server consolidation technique resulting best performance as it is initiated to solve server sprawl problem? What is its overall performance?

- Benchmark:

  How to analyze benchmark results in virtual environment? What is the relation between virtualization-level performance and application-level performance?

- Interference:

  How much deviation results on the performance of one virtual machine when it runs on the pool of VMs. How much impact is there?

The last option of the above points indirectly describes the scalability impact on the performance of virtual machine. Hence, this research is carried out to address the problem statement (section 1.2) by realizing the findings during literature survey, i.e., best results, best solution, best approach, best precaution and of course, with huge motivation.

1.5 Thesis Structure

The structure of the thesis is as follows:

Chapter 1 introduces a motivation for doing this research. Most related works

found during literature survey is also described in this section. Problem statement provides a clear vision on the main objective of this experiment. Under assumption and limitation, boundary of thesis is described.

Chapter 2 provides general overview of virtualization and its tools and techniques. The importance of benchmarking for this experiment, and different benchmarking methodologies are also described here. Fundamental concept of operating systems under test is given on the last section of this chapter.

Chapter 3 describes methodology for research. It explains the virtual environment by explaining virtual machine's creation, installation and configuration of benchmarking software. Important procedure during benchmarking, and statistical analysis approach to the extracted data is also described in this section.

Chapter 4 gives pictorial representation of different results obtained from the experiment. Showing the results in diagram makes easy to analyse the result.

Chapter 5 is entirely based on the above chapters. Review of approach for this experiment is explained here. Similarly, analysis of result and future work are also presented in this chapter.

Chapter 6 is final chapter. It summarizes the research.

Bibliography and Appendix: All the books, articles and other sources related to this research are given under bibliography, and the output of different types of benchmarking test are shown in the appendix.

Chapter 2

Background

Virtualization has long history and covers large area. It is not enough to explain the term virtualization by just few couples of lines. Its explanation may vary on different conditions, for example server virtualization, memory virtualization, I/O virtualization, network virtualization are different types of virtualization. In this thesis, virtualization refers server virtualization. Thus, only server virtualization and its related terms and technologies are explained in this chapter.

In the first part of this chapter, general overview of virtualization and its tools and techniques are presented. Second part focuses on benchmarking related terminologies. Last part gives brief introduction and basic architectural design of the operating system chosen for analysis.

2.1 Virtualization

Virtualization is one of the leading software technologies in the current computer industry. After introducing a concept of 'time sharing system' [49] by IBM in 1960s, it grew so rapidly. In the beginning, it was suitable for the mainframes and high end UNIX systems but after evolving x86 architecture [44] (Intel VT, AMD-V) with the ability of supporting virtualization extension, its popularity touch a new peak. Virtualization is a methodology of sharing hardware resources of a single computer into several instances of heterogeneous operating systems. It can create independent environment for each virtual machines on the top of virtualization layer i.e. hypervisor (section 2.1.3). Each virtual machine has its own virtual hardware and several applications can run

side-by-side on the same physical machine in the same time.

Along with innumerable benefits, server isolation and consolidation [46], live migration [2], dynamic load balancing [38], disaster recovery and high availability [40, 52], green computing [47], financial burden reduction, flexibility are major achievements of virtualization.

2.1.1 Virtualization Models

Server virtualization architectures are based on the hypervisor they used. Hypervisors are virtual machine Monitor which creates a layer of abstraction that isolates an operating system from underlying hardware. Hypervisors are responsible for managing virtual environment and handling large numbers of virtual machines running on same hardware. Figure 2.1 and 2.2 illustrate hypothetical diagram of different types of hypervisors. In general, there are two types of hypervisors, Type 1 and Type 2. Type1 hypervisor creates bare-metal virtualization architecture and Type 2 hypervisor creates hosted virtualization architecture.

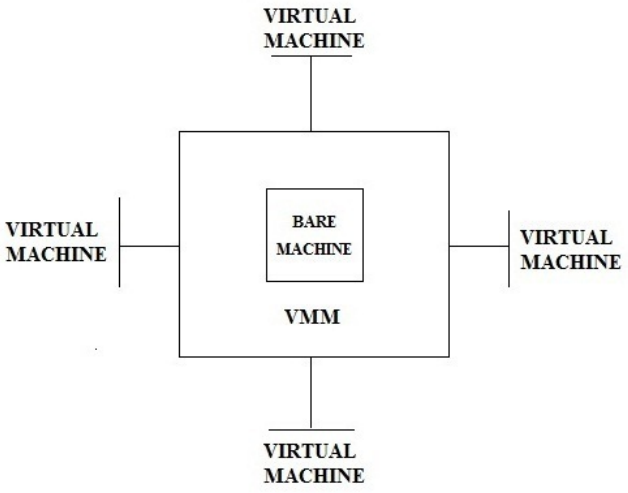

Figure 2.1: Hypothetical concept of type 1 hypervisor

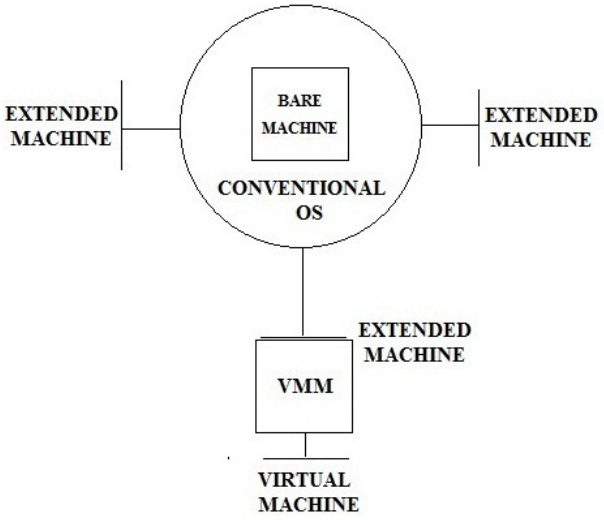

Figure 2.2: Hypothetical concept of type 2 hypervisor

2.1.1.1 Type 1 or Bare-metal Architecture

The hypothesis of this architecture (figure 2.1) is that virtual machine monitor runs on bare-metal [48]. It means that hypervisor runs directly on the top of given hardware platform (fig: 2.3) and can access it directly. In this architecture, virtual machine monitor is responsible for controlling the operating system environment by scheduling and allocating resources to all virtual machines running in the system. This hypervisor delivers high performance and better scalability.

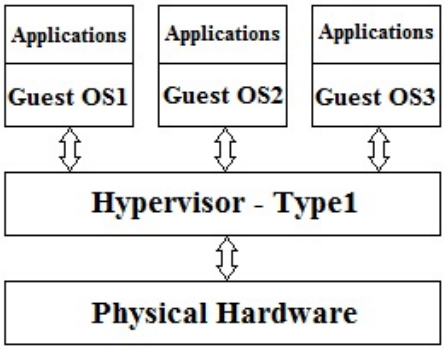

Figure 2.3: Architecture of Type 1 Hypervisor

2.1.1.2 Type 2 or Hosted Architecture

The principle of this architecture (figure 2.2) is that virtual machine monitor runs on extended host under the host operating system [48], which means that hypervisor runs as an application on the host operating system and guest operating system runs inside hypervisor (figure 2.4).

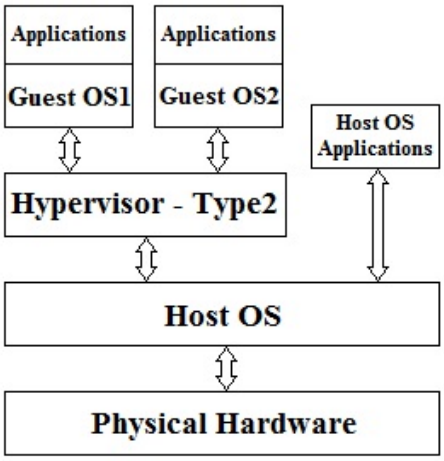

Figure 2.4: Architecture of Type 2 Hypervisor

Most important achievement by this approach is that several operating systems can be run on the top of the host operating system, and virtual machines can use all the hardware resources similar to the host OS does, but can not access directly. This concept may lead to security vulnerability in which guest OS gets full controls similar to the host OS has and also, it has performance penalty because both host OS and hypervisor have their own CPU manager and memory manager.

Most common types of hypervisors are described on section 2.1.3.

2.1.2 Virtualization Techniques

There are different forms of virtualization, which can be distinguished by their architectural layers. The most common approaches for virtualization are full-virtualization, para-virtualization and hardware-assisted virtualization. Full-virtualization and para- virtualization are software emulated virtualization, which have always been focused by users. This section gives a short overview

of these common techniques.

2.1.2.1 Full-virtualization Technique

In the late of 60s, the concept of full-virtualization was introduced by IBM research group by demonstrating it in CP/CMS operating systems, such as CP-40, CP-97. After then, this approach is carried out by IBM's VM family [24].

In this approach, hypervisor has higher privilege level than guest OS kernel, and it separates the operating system from the hardware resource logically. In full-virtualization technology [29], guest OS kernel does not need to modify hence it not only reduces managing efforts but also allows to create virtual environment using operating systems of close-source type. Unlike para-virtualization, it facilitates modularization by separating hardware and/or software into functional components thus customization of VM's setup is possible. Full-virtualization also offers secured migration [2] and delivers perfect isolation of guest OS from the underlying hardware so that its instance can run on both virtualized and non-virtualized conditions.

Unfortunately, its layered architecture (figure 2.5) possesses security management complexity and incurs performance penalty. Furthermore it delivers high system overhead, as it is responsible for caring all the system activity through the hypervisor.

2.1.2.2 Para-virtualization Technique

Para-virtualization is an old concept which exists since the decade of 1970s. In IBM's VM word, it was referred as Diagnose Code, and the Parallel Workstation calls it by a name of hypercall which means guests run a modified OS using a hypercall [14] [29]. In this architecture, hypervisor specific interface (figure 2.5) exists between virtual machines and virtualization layer to support important kernel operations and also helps to run hypervisor specific code during system call for making direct communication with hypervisor.

In general, It is a technique of changing the kernel of guest operating system to make it working under virtual environment. This concept was developed to ful-fill the drawback of full-virtualization and was used in Denali to host

Ilwaco operating system very first time to execute several untrusted and independent applications on a single computer [50] [51].

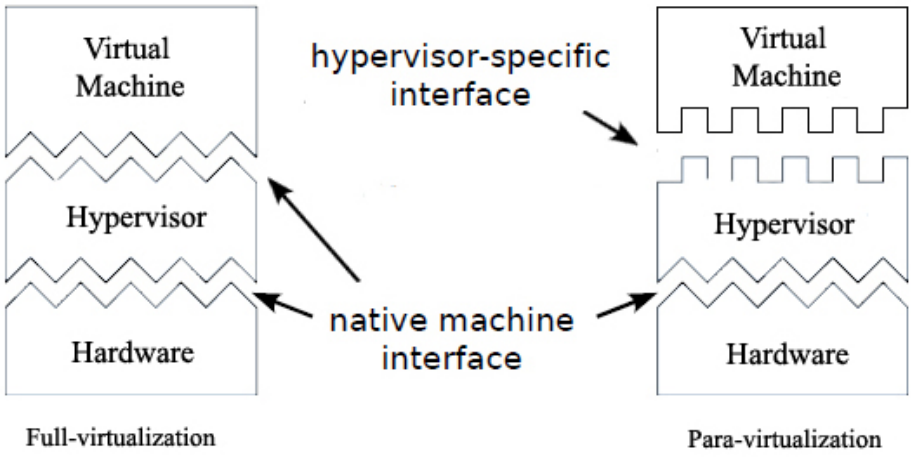

Figure 2.5: Comparison of Full-virtualization and Para-virtualization Techniques

The major limitation of this concept is that it lacks flexibility. It is because of the recompilation of guest operating system. This task is not only onerous but also difficult for closed-source operating system types. On the other hand, it achieves better performance than full-virtualization. It happens by saving time, as it does not care for trapping [45] privileged instruction, and no mediator is needed for making hardware call from the software running on VM's guest OS. This enriches para-virtualization to deliver best performance.

2.1.2.3 Hardware-Assisted Virtualization Technique

Hardware-assisted virtualization technique was introduced on IBM system in 1972. Later, Intel and AMD provided hardware support virtualization in 2005 [29]. Hardware-assisted virtualization also called as hardware virtualization. In this technique, virtual machine monitor moves into underlying hardware, and hardware supports to create a virtualization platform where multiple isolated virtual machines can run smoothly, but host computer system must have virtualization technology (VT) enabled hardware.

In this technology, CPU is virtually divided in to several virtual CPUs (vCPU) (figure 2.6). One vCPU is available for one virtual machines and respective

operating system runs on it. As a result, several operating systems can run parallelly in the same virtual environment without modification of guest operating system. At this step, hardware assisted technique works like a full-virtualization technique.

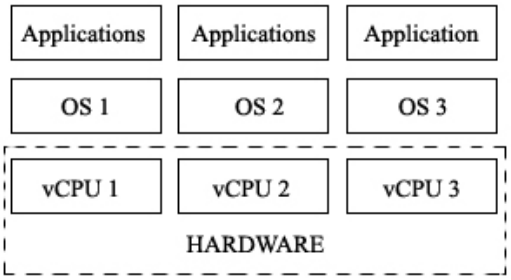

Figure 2.6: Hardware-assisted virtualization Techniques

Hardware-assisted virtualization does not need to modify guest operating system. As a result, extra maintenance effort is reduced, and it delivers better performance also. Hypervisor's overhead is eliminated in this architecture. On the other hand, this technique increases CPU overhead, and hence scalability of the virtual machine is limited. Similarly, VT enabled hardware are useful for this technique only.

2.1.3 Common Tools for Virtualization - Hypervisors

In the war of hypervisors, some virtualization tools are able to present them as leading virtualization technology. The most popular hypervisors are ESXi - a product of VMware, Inc., KVM - a product of Red Hat, Inc. and Xen - a product of Xen, Inc.. A brief overview of these tools is given in following sections. This research is done on KVM hypervisor.

2.1.3.1 KVM

KVM (Kernel Virtual Machine) is a latest generation of an open source hardware based full virtualization solution for x86 architecture supporting virtualization extension like Intel VT, AMD-V [36]. It was invented by Qumranet and supports execution of multiple virtual machines with unmodified guest operating systems like Linux and Windows. In KVM, the host operating systems must be Linux kernel version 2.6.20 or higher, as hypervisor capability is

integrated into host (Linux) kernel to result improved performance in virtual environment.

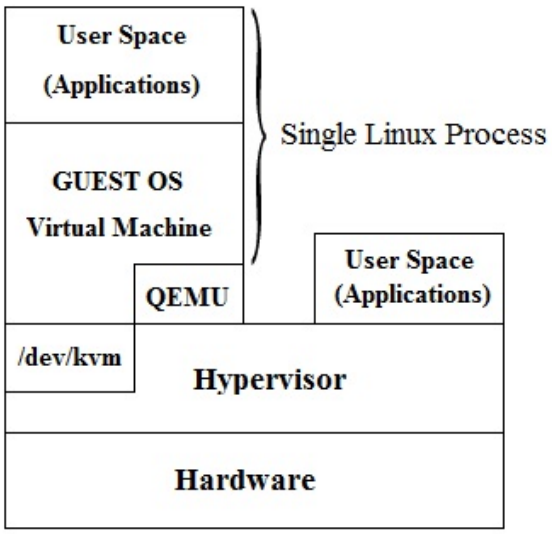

Figure 2.7: KVM Architecture

In KVM, Linux scheduler sets virtual CPUs as a regular Linux process used by VMs that permits KVM to share Linux kernel. Modified QEMU manages I/O service in user space [37]. Addition of a third mode in traditional way of process handling, i.e. a guest mode, simplifies process handling. Guest mode has its own user space and kernel space, which enables it to work as a single virtual machine. As a result KVM treats them as a single process (figure 2.7). KVM kernel is responsible for managing states of modes, making resources ready, assigning virtual CPU for virtual machine and so on.

2.1.3.2 Other Common Hypervisors

VMware ESXi

VMware ESXi, a modified face of VMware ESX hypervisor, is a virtualization tool offered by VMware Inc [32]. It is a bare-metal hypervisor which delivers high performance and supports large scalability. Instead of using Linux based service console, it uses remote command line interface resulting massive decrease in hypervisor codes. This creates a secure, highly reliable and simple virtual environment to manage.

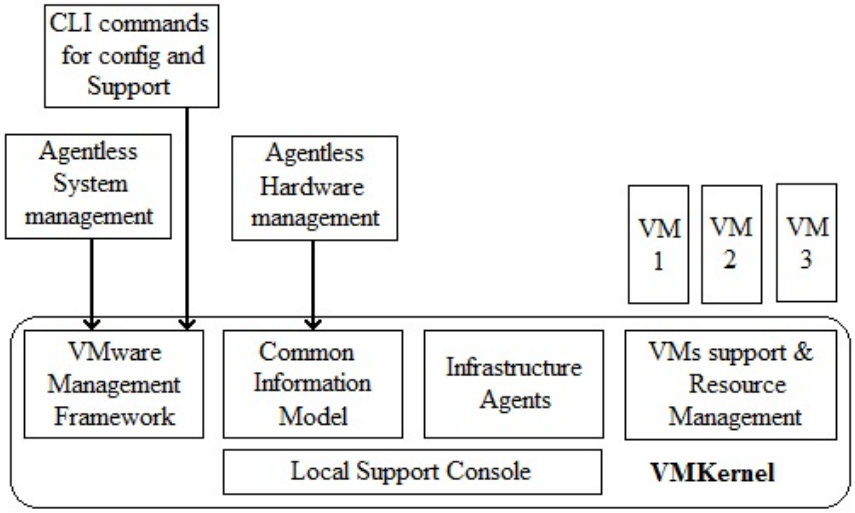

Figure 2.8: VMware ESXi Architecture

ESXi does not have service console [32]. Figure 2.8 shows that there is no service console in ESXi architecture. In ESXi, System and hardware management agents of ESX architecture are turned into agent-less approach. Management tasks are removed from hypervisor. Most common approaches for managing important tasks, such as user process and agent management can be done through vSphere Command Line Interface (vCLI) or powerCLI. They allow system administrators to monitor ESXi platform remotely. VMkernel is main component of VMware ESXi architecture which creates an environment to run several virtual machines, agents and other related management applications on the system [32, 35]. VMkernel is also responsible for resource scheduling, I/O stacks and device driver. VMware ESXi not only simplifies the configuration and installation but also makes easy to maintain the virtual environment.

Xen

Xen [36] is a open source virtualization standard developed by Cambridge University, UK. In general, it provides para-virtualization but needs virtualization enabled processor for full-virtualization. In Xen architecture (figure 2.9), virtual machines are considered as level 3 users so they can not access underlying hardware directly. A special virtual machine also called domain0 [42] is only allowed to access. It works as a host operating system and is responsible for managing virtual environment, as it has full access to the control

interface in Xen hypervisor.

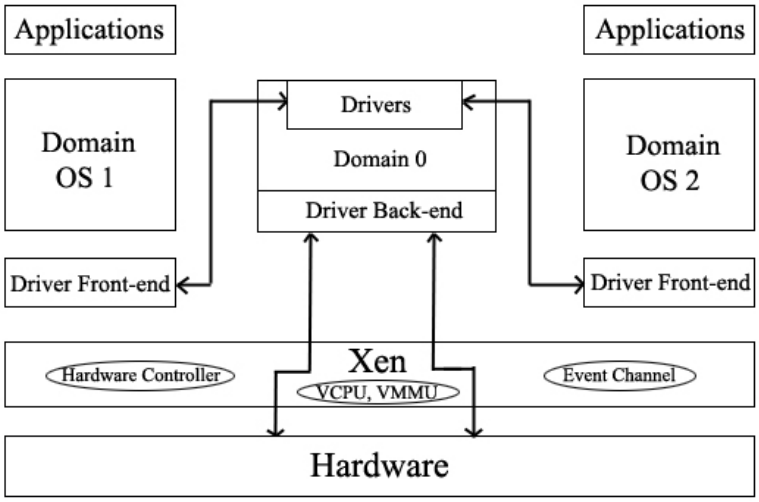

Figure 2.9: Xen Architecture

Device manager and control software running on domain0 help to manage common processing environment of multiple virtual machines. It also allows domain0 to access I/O resource and to communicate with virtual machines through device drivers [42]. Virtual machines are given to access resources provided by domain0 interface only. Actually, guest domains (VMs) access drivers via backend drivers provided by domain0. In addition, Xen also allows to create special privileged virtual machines that have direct access to resource through its secure interface.

2.2 Benchmarking

In computer system, simple prediction of performance of objects and/or applications by looking their working style and their labeled specification does not lead for strong decision making. Correct evaluation of device performance, computational performance, network performance and protocols is essential to judge overall system status. Benchmark tools facilitate the testing of these performances. Benchmark tool is a well structured piece of program designed to evaluate performance of objects and their work load on the system [39].

Generally in computing, benchmarking is a standard process of evaluating and measuring the performance of object under different conditions. A benchmark

must have four major specifications to test the system performance [41]. They are scenario, evaluation criteria, evaluation metrics and benchmark score. A single test cannot reflect the whole system performance and also it is very crucial to know the testing purpose and its suitable tool. For example, the output of one benchmarking - suppose the benchmark result of disk performance test does not have sense to predict the performance of CPU. In the following sections, these matters are further discussed.

2.2.1 Requirements for Benchmark

A statistics provided by benchmark is very useful for decision making to system administrator and/or organization. So during the development of benchmarking tools they must be well programmed. Benchmark is all about "understanding a complex system" [43]. It not only directs for the best system designing by identifying bottlenecks, but also creates a best platform to compare two or more systems or components.

For a reasonable statistics and optimum accuracy, benchmark must have some major characteristics.

- Comparability:

  It is basic property of benchmark. Benchmark result must be comparable with other result to choose the best and suitable one.

- Repeatability:

  Benchmark must be repeatable so that benchmarking on different phase of time can give an identical result.

- Configurability:

  Benchmark must be configurable. In some test environments, benchmark need to configure for better functionality.

2.2.2 Benchmarking Classification

Classification of benchmark is usually done on the basis of metrics they use or their workload on the system. If it is classified on the basis of performance level, it can be categorized in two types:

- System-level Benchmark:

  System-level benchmark determines the overall performance of the system, which is done by determining the behavior and effect of each subsystem that are running inside the system under test. Sometimes system-level benchmark is referred as high-level benchmark also.

- Component-level Benchmark:

  Component-level benchmark is designed to measure the performance of specific object or component of the system. It is also known as low-level benchmark.

But, if it is classified based on their composition, it falls on another two types:

- Application Benchmark:

  Application benchmarking is a better way of measuring computer system performance. It can present overall system performance by testing the contribution of each component of that system. Thus, application benchmark is also considered as system-level benchmark and sometimes as a macro-benchmark.

- Synthetic Benchmark:

  Synthetic benchmark is more flexible than application benchmark [18]. It was developed to measure the performance of individual component of computer system. So micro-benchmark is considered as a derivation of synthetic benchmark also. It uses large number of parameters which helps to increase the number of workload and also supports for scalability.

On the basis of benchmarking classification mentioned above, there are three basic categories for performance analysis of operating system. They are:

- Macro-benchmarking
- Micro-benchmarking
- Kernel Profiling

2.2.2.1 Macro-benchmarking

Macro-benchmark shows entire performance of computer system by measuring the overall performance of actual applications on a specific workload by comparing its application class. Macro-benchmarking is closely related with the application specific benchmarking. It is also capable of finding non-kernel bottlenecks [57], but its major drawback is that it always needs middleware to show the results.

2.2.2.2 Micro-benchmarking

Micro-benchmarking is focused on one particular component in the system, such as CPU clock, disk access time, latency, throughput, memory speed etc., and thus are unable to show holistic performance of the system. Micro- benchmarking concerns for determining the weakness and the strength of particular object, that is why it just reflects low-level behavior of the system. Usually this technique concerns on the OS kernel's abstraction layer, mainly focusing for identifying impact of patch on the kernel subsystem.

2.2.2.3 Kernel Profiling

Kernel profiling benchmarking is the best procedure for measuring the behavior of kernel procedures, and its time spent during individual kernel procedures. Tracing [55] is one major procedure in this benchmarking by which testing program generates a trace of flow control of events. Similarly, statistical profiling [56] is another mechanism by which states of the processor, such as pointer, address space identifier etc. are stored and later, statistical summary is displayed. The fundamental problems with this benchmark are that they are usually small, fit in cache, are prone to attack by compilers and measures only CPU performance.

2.2.3 Benchmarking and this research

As described in section 1.2, the main scope of this research is to determine the impact on virtual machine on the presence of other virtual machines. Impact on virtual machine can be determined by its performance degradation. To observe a system for addressing given problem statement, benchmarking is only solution.

Benchmarking tools determine the system performance in low-level and in high-level. A hybrid approach, a combination of low-level and high-level benchmarking; low-level benchmarking, like CPU latency and bandwidth and high-level benchmarking, like application workload on the system, is the best approach of performance analysis of the computer system. In this research, the purposed virtual machine is considered as a white box for micro-performance analysis and as a black box for macro-performance analysis, where different benchmark tools are used on the basis of their easiness to configure, sufficient documentation and broad categories of their result. Generally, benchmark tool evaluates the system on the basis of work load on the system and application services running in that time. It does neither concern on architecture nor protocols used by the system. This research is conducted in pre-determined hardware infrastructure (see 3.4). Each phase of experiment is conducted on the same architectural design. This means that multiple virtual machines run on the same hardware resource which creates a fundamental background to evaluate the performance of the guest systems running on it.

2.3 Operating System

The term 'operating system' is well known among the computer users. In simple way it can be defined as a set of program which controls the computer. In technical way it can be explained as a set of programs which has ability to communicate with the system's hardware to allow other applications to run. It is also responsible for process management, memory management, file management, I/O management, Disk management, network management and so on.

Up to now, large numbers of operating systems have been developed, but

Windows and UNIX are two major classes of operating systems. Although having common goal, these two classes of operating systems have completely different concept to each other. Windows Vista and Windows 7 are examples of Windows OS. Debian, Ubuntu, Mac, Fedora are UNIX based operating system. This thesis does not cover all these and only concerns on Debian GNU/Linux because it is a purposed operating system for this research.

2.3.1 Debian GNU/Linux

Debian GNU/Linux, in short Debian, is an open source operating system. The concept was initiated in 1993 by a name of 'Debian Linux Release' but first 1.x version was released in 1996. Debian uses tools from GNU operating system and its kernel is based on Linux kernel. Hence, it is called as GNU/Linux.

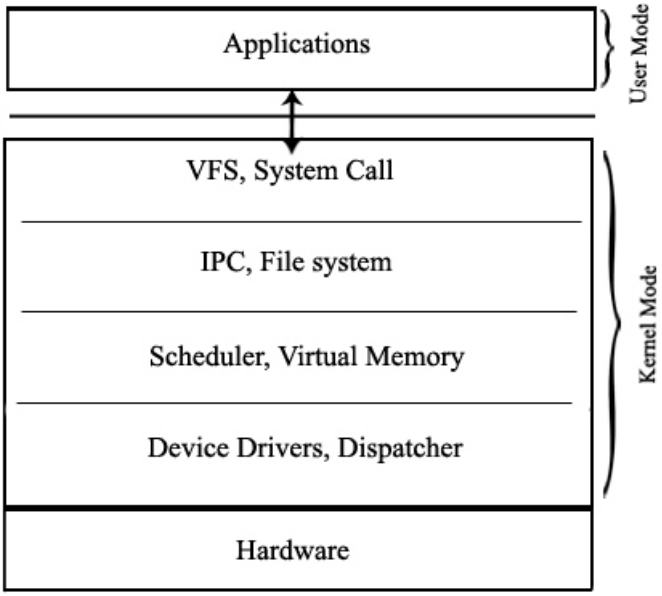

Figure 2.10: Structure of Monolithic Kernel

Debian is based on monolithic kernel architecture (figure 2.10) where entire operating system works in kernel mode. All the system processes are a part of kernel and run on ring 0. In this architecture, its kernel extension and device driver have full access to the hardware. The most prominent feature of package management facility has distinguished Debian as a perfect operating system among other Linux based operating system. This allows full control to a privileged computer user over software packages in the system. Currently, De-

bian supports eleven architectures for example: x86-32, x86-64, PowerPC etc. In each version, Debian is adding extremely good features such as graphical user interface, JAVA friendly compilers, security and so on. Debian Squeeze is its latest version. It is one of the most popular and leading operating system among Linux-based operating systems.

Chapter 3

Methodology

In this research, the performance of the virtual machines is determined on two phases. In first phase, the targeted system is considered as a white box and in next phase, it is considered as a black box. White box testing is used to determine the micro performance of targeted system, where internal working mechanism of system is analysed. This experiment figures out the latencies and bandwidth of system operations and context switching of purposed system. Generally, micro benchmarking can not reflect the overall system performance. It only reflects low-level system operations. Hence, macro performance testing is essential. In the next phase of testing, the system is considered as a black box. Unlike white box testing, internal mechanism of system is not considered. The result is analysed on the basis of output of the system. In this phase, macro performance of the system is determined by examining memory, CPU, network and disk operations. Finally, the analysis is done on basis of micro benchmarking and macro benchmarking result.

In the following sections, different components of virtualization environment, steps of benchmarking procedure with its related tools and their purpose are described.

3.1 Describing Environment

The term 'environment' refers to virtualization environment. Different tools and techniques (hardware and software) are used to create this environment. Figure 3.1 illustrates an example condition of this experiment, where 2 virtual machines are up and remaining 12 machines are down.

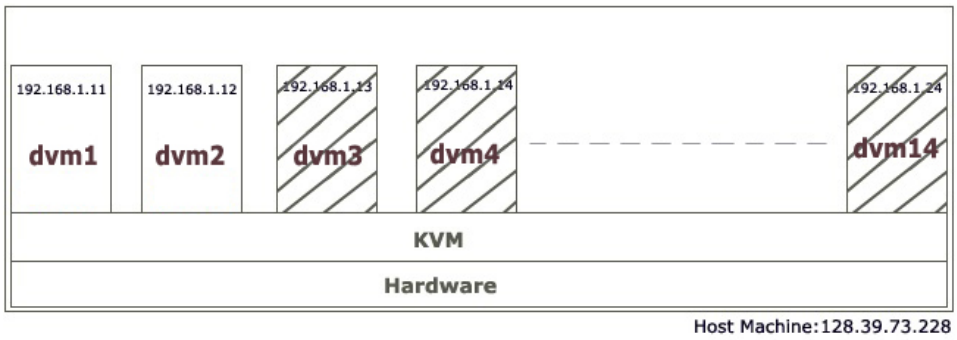

Figure 3.1: Virtual Environment of this Research

Hypervisor: Kernel-based Virtual Machine (KVM)

As described on section 2.1.3.1, KVM is a one of the most common hypervisor these days, which is designed as a kernel module and added into Linux kernel. It provides full-virtualization facility.

Operating Systems

In this experiment, Debian GNU/Linux (see 2.3.1) is a purposed operating system. In both phase of testing (macro and micro benchmarking), Debian is installed on all virtual machines. It means Debian is running on all virtual machines as a guest operating system.

Virtual Environment: Proxmox

Proxmox is a Linux Debian based framework to create virtual environment. With its graphical user interface and powerful web-based control panel, management of virtual environment is easier. It facilitates creating virtual machines based on both OpenVZ and KVM hypervisors, and facilitates clustering and live migration also. Screen-shot of Proxmox is shown in figure 3.2. It is an open-source software and can be downloaded easily from http://pve.proxmox.com/wiki/Downloads. In this experiment before installing guest operating system, iso image of Debian operating system is uploaded in Proxmox. Iso image of Debian OS is downloaded from ftp://ftp.uio.no/debian-iso/.

3.2 Benchmarking Tools

Performance analysis of the system is done on the basis of benchmarking output. Several benchmarking tools can be found in the Internet. There is not any criteria to choose tools and software, but it depends on what is going to be analysed and what is a goal of the research. Depending on the problem statement of this research, sufficient documentation, easy installation, configuration and a broad range of output, following benchmarking tools are chosen for this research.

3.2.1 Lmbench

Lmbench is a micro benchmark suite that was developed by Larry McVoy and Carl Staelin and written in ANSI-C. This benchmark was developed to test important aspects of system performance like latency, bandwidth, clock speed and instruction-level parallelism [30, 31].

Type	**Description**
Bandwidth	file re-read using read() and mmap(), IPC using TCP, pipe, and unix sockets.
Latency	memory latency, TCP and unix socket connection, IPC using TCP, UDP, RPCs, pipe, and unix sockets, file creation and deletion, process creation using fork(), fork()+exec(), and sh(), select() on file descriptors and network sockets, mmap(), page faults, signal installation and handling, system calls, context switching, basic arithmetic operations on various data types, and overall time to complete N jobs which each do micro secs-worth of work.
Other	CPU clock speed, cache line size, TLB size, basic operation instruction-level parallelism, memory subsystem instruction-level parallelism, and STREAM benchmarks.

Table 3.1: Different types of test with Lmbench

This benchmark concerns on low-level kernel primitives of operating system. It generates a board range of output but only important outputs that are related to this research are chosen, which are shown in table 3.3.

Type	Description
Processes Latencies - fork proc - exec proc - sh proc	Processes are fundamental unit of work on the system. Process latencies represents overhead of process creation, such as fork duplicates process and exec overwrites process. They are measured in micro seconds, and smaller latency is considered as better.
Communication Latencies - Pipe Latency - TCP Latency - UDP Latency	It is interprocess communication latency where 1 byte message is rotated between two processes, and the latency is calculated on microseconds. Only pipe latency, tcp latency and udp latency are measured in this experiment.
Memory Bandwidth - Memory Read - Memory Write - bcopy (libc)	Memory read is time to read data into the processor. Memory write is time to write data into memory. The bandwidth of memory read and write is calculated in mbps. Bcopy measures how fast a system can bcopy the data.
Communication Bandwidth - TCP Bandwidth	It is reading of data via TCP socket. Lmbench determines it as a bandwidth of local TCP stack. Bigger bandwidth is considered as better.
File System Latency - File Create - File Delete	Here, file system latency is defined as a time required for creating and deleting large number of files, almost 1000 files. This benchmark performs test on 0k sized and 10k sized file. This is measured on microseconds.
Memory Latency	It measures memory read latency for varying memory sizes and strides. Smaller is considered as better.
Context Switching	It is a switching time between processes where current running processes are saved, new processes are loaded and their executions are began. Context switches can occur only in kernel mode. It is measured on microseconds. If a system has least context switching time, it is considered as best system.

Table 3.3: Purposed Lmbench Test Detail

3.2.2 Bonnie++

Bonnie++ is a disk IO benchmarking tool and is based on bonnie. Bonnie++ was developed by Russell Coker. It performs series of tests on the file system and presents the results in two different parts. Each part consists of six test. First part is focus on IO intensive application which is determined by doing different test on given size file, and the result is shown in number of kilobytes processed per second. The second part focus on file creation test by doing sequential creation and random file creation tests. In both parts, the percentage of memory usage during a particular test is shown also.[23]

All the details of file I/O test of bonnie++ is shown in table 3.4.

Sequential Output	- Per character : using putc() to write single character - Per block : block output using write() - Rewrite : using read, write and reread
Sequential Input	- Per character : using getc() to read single character - Per block : reads block of data from file using read()
Random Seek	- runs SeekProcCount processes in parallel. It determines how good system can order seeks and how fast system hardware can do random accesses

Table 3.4: Bonnie++ first six tests - File I/O test

All the details of file creation test of bonnie++ is shown in table 3.5.

Sequential Create	- Creates files in sequential order - Stat files in sequential order - Deletes files in sequential order
Random Create	- Creates files in random order - Stat files in random order - Deletes files in random order

Table 3.5: Bonnie++ second six tests - File creation test

The major options used for bonnie++ are as follows;

-d = directory for testing
-s = size of files for test (usually double of RAM size)
-n = number of files for file creation test

-u or -g = run as user or group
-p = number of processes used by semaphores.

When bonnie++ runs, it reads, writes and deletes files on disk. The disk performance can be viewed from */proc/diskstats*. This directory contains a lot of information and looks like:

```
   8       0 sda 1705 345 52174 1232 1247 464 13704 63120 0 4400 64352
   8       1 sda1 1553 314 50722 1188 1247 464 13704 63120 0 4376 64308
   8       2 sda2 2 0 4 4 0 0 0 0 0 4 4
   8       5 sda5 102 31 1064 12 0 0 0 0 0 12 12
  11       0 sr0 19 0 152 12 0 0 0 0 0 12 12
   7       0 loop0 0 0 0 0 0 0 0 0 0 0 0
   7       1 loop1 0 0 0 0 0 0 0 0 0 0 0
   7       2 loop2 0 0 0 0 0 0 0 0 0 0 0
   7       3 loop3 0 0 0 0 0 0 0 0 0 0 0
   7       4 loop4 0 0 0 0 0 0 0 0 0 0 0
   7       5 loop5 0 0 0 0 0 0 0 0 0 0 0
   7       6 loop6 0 0 0 0 0 0 0 0 0 0 0
   7       7 loop7 0 0 0 0 0 0 0 0 0 0 0
```

There are 11 fields in */proc/diskstats*. Each field provides different information. They are as follows:

```
Field 1 – # of reads issued
Field 2 – # of reads merged
Field 3 – # of sectors read
Field 4 – # of milliseconds spent reading
Field 5 – # of writes completed
Field 6 – # of writes merged
Field 7 – # of sectors written
Field 8 – # of milliseconds spent writing
Field 9 – # of I/Os currently in progress
Field 10 – # of milliseconds spent doing I/Os
Field 11 – weighted # of milliseconds spent doing I/Os
```

3.2.3 NetIO Benchmark

NetIO is a network benchmark tool for Windows and Unix operating system, and measures the network throughput via TCP and UDP protocols using various packet sizes (1k, 2k, 4k, 8k, 16k and 32k). In this test, one machine serves as a server and other machine acts as a client. When server receives packets

from the client, it send back to client, and meanwhile network throughput is determined. The major options used for NetIO are as follows;

-s = Work as a server
-b = Block size in kbytes (without specifying it uses 1,2,4,8,16 and 32k)
-p = Port number (default is 18767)
-u = Use UDP protocol for benchmark
-t = Use TCP protocol for benchmark
(server) = Server name or IP (only for client machine)

For example, to run NetIO bechmark in server machines, *./linux-x86_64 -s -p 18767 -t* command is supplied. It means the system is working as a server, listening on port number 18767 and using TCP protocol for throughput test. Similarly, a command in client machine, ./linux-x86_64 -t -p 18767 128.39.73.228 means that the system is working as client, using TCP protocol, requesting service on the port 18767, and IP of server is 128.39.73.228. When a particular phase of testing finishes, the output is generated. A sample output is shown in Appendix C.

3.3 Security of Experiment

This experiment is conducted on a machine having public IP. It means the machine under test is directly accessible from the outside of the university. To reduce risk from probable malicious activity, a firewall rule is implanted on the host machine. So that users from limited IP range are only allowed to access the machine. The firewall rule is given here Appendix D.

3.4 System Specification

3.4.1 Host Machine

In this experiment, one physical computer is taken as a host machine. This is a main machine where large number of virtual machines are created. The hardware specification of this machine is shown in table 3.6.

Type	Description
Processor	Quad-Core AMD Opteron(tm) Processor 2376, Total: 8, 2.30 GHz, Bus Speed 1000 GHz, 2MB L2 Cache, 6MB L3 Cache, HT 1 Technology
Memory	32.0 GB of 667 MHz, ECC DDR2 Type
Hard Drive	500GB - ATA device, Nominal Media Rotation Rate: 7200
I/O Ports	4 USB ports, 2 network interface, 1 VGA port, 1 DVD, 1 DVI port
Software	Kernel: 2.6.32-6-pve GNU/Linux, x86_64, OS Distributor ID: Debian, Release: 6.0.5, Codename: squeeze
VT	Virtualization Technology is enabled

Table 3.6: Host Machine Specification

3.4.2 Virtual Machine

Each virtual machine is created with exactly similar configuration. Hence, all virtual machines are identical to each others. The system information of one VM is given in table 3.7.

Type	Description
Processor	1 Virtual CPU from host machine
Memory	Virtual 2GB RAM
Hard Drive	Virtual 32GB HDD, SCSI Type
Networking	e1000 Network Driver

Table 3.7: Virtual Machine Specification

3.5 Approach

3.5.1 Creating Virtual Machines

Proxmox is used to create larger number of virtual machines, here 14 machines. Its secure web-based control panel provides extremely easiness to manage scalable virtual environment. A screen-shot of creating one of the virtual machine is shown in fig 3.2.

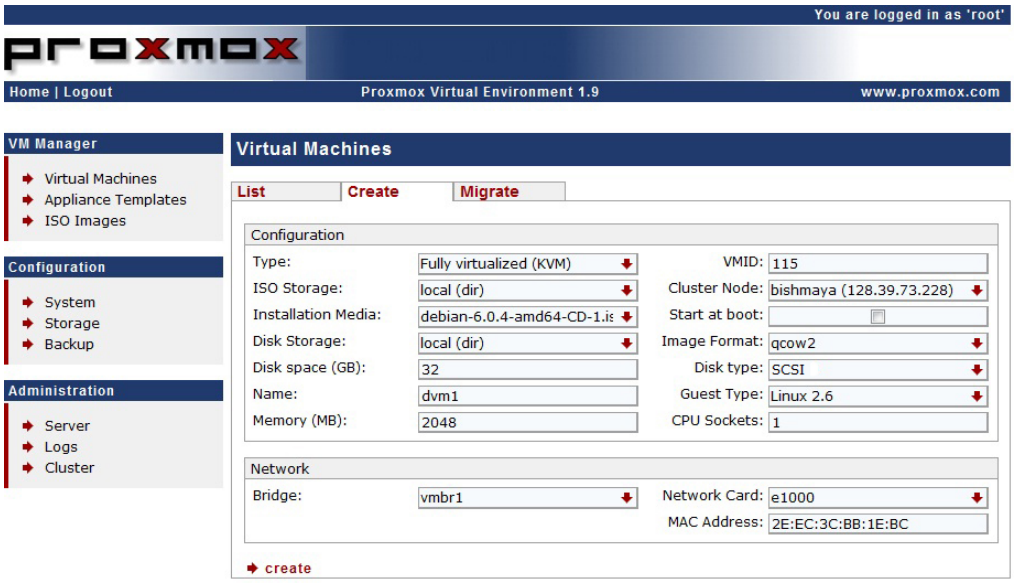

Figure 3.2: Proxmox screen-shot of creating virtual machine

3.5.2 Procedure

After creating 14 virtual machines, Debian operating system is installed on all virtual machines one-by-one. Each operating system is installed with minimal package to avoid unnecessary system resource consumption.

White-box Testing

1. After setting up a complete environment, **micro-benchmark** tool, lmbench, is installed on all virtual machines. Installation and configuration procedure is described on 3.6.1.

2. All virtual machines (dvm2 to dvm14) are then turned-off except first Debian virtual machine (dvm1). Then, lmbench is executed on dvm1 and data is collected.

3. To make experiment more accurate and representative, the experiment is repeated (rerun) 3 times.

4. All the results are stored on appropriate location which are further analysed statistically. Only those data which are more related to the problem statement of this research are extracted from the result, and mean (see 3.7) is calculated to represent a solo figure of that particular test. With

the help of Microsoft Excel Tool, the final output is generated on graphs. The pictorial representation of result is shown in section 4.1.

5. Next, dvm2 is turned-on. Lmbench is executed on both virtual machines and above procedures described on points '3' and '4' are repeated. As this experiment is concerned on the performance degradation of first virtual machine, the output of lmbench is collected in dvm1 only.

6. Similarly, dvm3, dvm4 upto dvm14 are turned-on and, lmbench is executed on each virtual machine one-by-one. Above steps described on points ('3' and '4') are repeated, and output is collected on dvm1 only.

After completing micro-benchmark test, **macro-benchmarking** is conducted on the system. In this test, system is considered as a **black box** where internal mechanism of system is not considered.

Disk Performance Test

1. Before starting disk performance test, all unnecessary packages are removed from all the systems and disk performance testing tool, bonnie++, is installed on all virtual machines. Installation and configuration procedure is described on 3.6.2.

2. All virtual machines (dvm2 to dvm14) are then turned-off except first Debian virtual machine (dvm1). Then, bonnie++ is executed on dvm1. In this research, disk test is done on little bit different way. Generally when bonnie++ runs, it reads and writes on specified disk which can be viewed via *'cat /proc/diskstat'* command. To understand reading and writing sequence on disk, a small perl script is developed (see 3.6.2). In each 5 seconds interval of time, this scripts collects data of 'sectors read' and 'sectors write' from *'/proc/diskstats'* during bonnie++ execution and stores on a separate file for further analysis. On the other hand, bonnie++ also generates result (see Appendix B) itself which is stored separately.

3. To make experiment more accurate and representative, the experiment is repeated 3 times.

4. All the results are stored on appropriate location which are further analysed statistically. Only those data which are more related to the problem statement of this research are extracted from the result, and mean (see

3.7) is calculated to represent a solo figure of that particular test. With the help of Microsoft Excel Tool, the final output is generated on graphs. The pictorial representation of result is shown in section 4.2.1.

5. Next, dvm2 is turned-on. Both bonnie++ and the script are run on both virtual machines, and above procedures described on points ('3' and '4') are repeated. As this experiment is concerned on the performance degradation of first virtual machine, the output of perl script and bonnie++ are collected in dvm1 only.

6. Similarly, dvm3, dvm4 up to dvm14 are turned-on. Bonnie++ and the script are executed on each virtual machine one-by-one. Above steps described on points ('3' and '4') are repeated and output is collected on dvm1 only.

Processor Performance Test

Processor virtualization overhead is determined using a simple Linux command '*mpstat*', small perl script (load.pl, see Appendix E) as a stress generator in CPU and another perl script (capture.pl, see below) to capture the results. The capture.pl is a special script which stores the system activity information by executing *mpstat* command in each 5 second interval of time.

1. Before starting processor performance test, all unnecessary packages are removed from all the systems and a Linux package 'sysstat' is installed in all virtual machines using *apt-get install sysstat*.

2. All virtual machines (dvm2 to dvm14) are then turned-off except first Debian virtual machine (dvm1). Then, load.pl and capture.pl are run in the same time for 10 minutes.

3. To make experiment more accurate and representative, the experiment is repeated 3 times.

4. All the results are stored on appropriate location which are further analysed statistically. Only those data which are more information are extracted from the result, and mean (see 3.7) is calculated to represent a solo figure of that particular test. With the help of Microsoft Excel Tool, the final output is generated on graphs. The pictorial representation of result is shown in section 4.2.4.

5. Next, dvm2 is turned-on. Both scripts are run on both VMs and, above procedures described on points ('3' and '4') are repeated. The output of capture.pl is collected in dvm1 only because this experiment is concerned on the performance degradation of first virtual machine.

6. Similarly, dvm3, dvm4 up to dvm14 are turned-on one-by-one and, both scripts are run on all VMs. Above steps described on points ('3' and '4') are repeated and output is collected on dvm1 only.

```
#! /usr/bin/perl
open(FILE1, ">>/root/usr.txt") or die ("Can not Create File.\n");
open(FILE2, ">>/root/sys.txt") or die ("Can not Create File.\n");
open(FILE3, ">>/root/idl.txt") or die ("Can not Create File.\n");
while(1)
{

   @bis = qx(mpstat);
   if( $bis[3] =~ /(.+)\s+(.+)\s+(\d+.\d+)\s+(\d+.\d+)\s+(\d+.\d+)\s+(\d+.\d+)
       \s+(\d+.\d+)\s+(\d+.\d+)\s+(\d+.\d+)\s+(\d+.\d+)\s+(\d+.\d+)/ )
   {
    print FILE1 "$3\n";
    print FILE2 "$5\n";
    print FILE3 "$11\n";
   }
  sleep 5;
}
```

Network Performance Test

NetIO throughput benchmark is chosen for network performance test of the system. Installation and configuration procedure of NetIO is described on section 3.6.2. In this test, all virtual machines works as a server for TCP throughput test, and works as a client for UDP throughput test. The only concern of making system in different forms (server / client) is not more than making a real world scenario. A separate Linux virtual machine (128.39.73.234) provided by friend is made client for TCP throughput test. NetIO is installed on client machine also. In this test, server always stays in listening mode and replies back to the client when it gets traffic from the client. Before starting test, some lines of firewall rule (see Appendix D) is added for port forwarding. So that proxmox machine (host machine) can forward the ports to respective port of targeted machine where NetIO service is available. A small perl script is written, which can execute NetIO commands from the client machine exactly

on the same time to different servers. The script also can store the output of NetIO command. Following points describes the method of network performance testing.

1. NetIO is installed on every machines and then all virtual machines (dvm2 to dvm14) are turned-off except first Debian virtual machine (dvm1). Then, the server system is set in listening mode (./linux-x86_64 -s -p 18767 -t) so that it can reply the request coming from client machine.

2. From the client machine, 1_up.pl script (see below) is run for sending traffic (packets) from client to server (dvm1) in specified time, and the reply of server is recorded. Example output format : see Appendix C

3. To make experiment more accurate and representative, the experiment is repeated 3 times.

4. All the results are stored on appropriate location which are further analysed statistically. And mean (see 3.7) is calculated to represent a solo figure of that particular test. With the help of Microsoft Excel Tool, the final output is generated on graphs. The pictorial representation of result is shown in section 4.2.3.

5. Next, dvm2 is turned-on. To send the packets from client to both the servers, different perl scripts (1_up.pl and 2_up.pl) are run on the client, which send packets to the servers, dvm1 and dvm2, exactly on the same specified time respectively. Then, above steps described in point '3' and '4' are repeated.

6. Similarly, dvm3, dvm4 up to dvm14 are turned-on one-by-one and different perl scripts are run on the client machine in accordance with the running servers. Above steps described on points ('3' and '4') are repeated and output is collected in dvm1 only.

7. For UDP throughput test, all the 14 virtual machines are made clients and remote machine (s160487@studssh.iu.hio.no) is made server. All the previous steps described from points 1 to 6 are repeated.

```
##### 1_up.pl :- To send packet from client to dvm1. #####

#! /usr/bin/perl
```

```
open(FILE, ">>/home/group3/tcp_netio_result.txt") or die ("Can not Create File
    .\n");
while(1)
{
    @timeData = localtime(time);

    if($timeData[1]==52)
    {
        @bis = qx(./linux-x86_64 128.39.73.228 -p 4021 -t);
        print FILE "@bis";
        exit;
    }
}
close (FILE);

##### 2_up.pl :- To send packet from client to dvm2. #####

#! /usr/bin/perl

while(1)
{
        @timeData = localtime(time);
    if($timeData[1]==52)
                {
                qx(./linux-x86_64 128.39.73.228 -p 4022 -t);
            exit;
        }
}
close (FILE);

# N.B. To send packet from client to dvm3 and so on, only port number is
    changed and same script is used. The firewall rule written in proxmox
    machines forwards the the client request coming on port 4021 to port 18767
     of dvm1, request coming port 4022 to port 18767 of dvm2, request coming
    port 4023 to port 18767 of dvm3 and so on.
```

Above steps are a complete procedure of micro-benchmark and macro- benchmark test of Debian guest machines which are running on KVM based virtual environment.

3.6 Configuration

3.6.1 White Box Testing

Lmbench: Installation and configuration

Lmbench, a micro-benhmarking tool is downloaded from its home page (http://www.bitmover.com/lmbench/get_lmbench.html) which is then unzipped

from compressed format.

Before executing lmbench, all necessary compilers are installed with *apt-get install build-essential* command.

To start execution, *make results see* command is run from the root folder of lmbench. All required input is supplied during its execution. When it accepts all related input, actual benchmarking begins, and after some time it confirms the completion of execution with following output. Execution time may vary according to system processing strength.

```
Using config in CONFIG.vm1
Sat Mar 31 20 20:33:35 CEST 2012
Latency measurements
Sat Mar 31 20:34:34 CEST 2012
Calculating file system latency
Calculating disk zone bw & seek times
Sat Mar 31 20:34:36 CEST 2012
Local networking
Sat Mar 31 20:35:54 CEST 2012
Bandwidth measurements
Sat Mar 31 20:38:31 CEST 2012
Calculating context switch overhead
Sat Mar 31 20:38:35 CEST 2012
Calculating effective TLB size
Sat Mar 31 20:38:36 CEST 2012
Calculating memory load parallelism
Sat Mar 31 20:50:42 CEST 2012
McCalpin's STREAM benchmark
Sat Mar 31 20:51:44 CEST 2012
Calculating memory load latency
Sat Mar 31 21:14:44 CEST 2012
make[1]: Leaving directory '/root/lmbench/lmbench-3.0-a9/src'
```

As described on previous section, each phase of test is repeated 3 times. Hence in second time test, *make rerun see* command is supplied instead of *make results see*. This omits re-supply of user input during execution of lmbench. And result is generated on the basis of user input supplied in the beginning of its first test.

Result of lmbench can be viewed from *summary.out* file (see Appendix A) located in */lmbench/bin/results* directory. More specific information of *summary.out*

can be found in */lmbench/results/os_name/host_name*.

3.6.2 Black Box Testing

Bonnie++: Installation and configuration

Bonnie++, a file system benchmarking tool, is installed on the system using *apt-get install bonnie++* command.

Bonnie++ is very easy to use. Just a single line command (bonnie++ -u *user* -d *directory* -s *size*) is enough to execute it. In this experiment, following bonnie++ command is supplied:

bonnie++ -u root -d /root/bonnie_test/ -s 5G

During its execution, a simple perl script is also run on background which collects data of 'sectors read' and 'sectors write' from '/proc/diskstats' in 5 second interval of time.

```
#! /usr/bin/perl

open(DISK, "/proc/diskstats") or die ("Error.\n");
open(FILE1, ">>/root/ result/up10_field3.txt") or die ("Error.\n");
open(FILE2, ">>/root/ result/up10_field7.txt") or die ("Error.\n");

while(1)
{
    $line = qx(grep "sda" /proc/diskstats);
    if ($line =~ /\s+(\d+)\s+(\d+)\s+(\w+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s
        +(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)/ )
    {
        print FILE1 "$6\n";        # storing data in separates file makes easy
             to analyse
        print FILE2 "$10\n";
    }
    sleep 5;
}
close (DISK);
close (FILE1);
close (FILE2);
```

On the other hand when bonnie++ completes execution, it produces a result itself. A sample output is shown in Appendix B.

NetIO: Installation and configuration

NetIO benchmark is very easy tool to implement. The zip file of NetIO is download from its home page using *wget http://www.ars.de/ars/ars.nsf/f24a6a0b94c22d82862566960071bf5a/aa577bc4be573b05c125706d004c75b5/$FILE/netio131.zip* command and file is then unzipped after download.

After extraction, a */bin* directory can be seen where several binaries files are stored. According to system architecture of this research, a file named linux-x86_64 is made executable which is used to test network throughput. During the execution of linux-x86_64 file, -s options represents the system is working as server otherwise system is client.

3.7 Calculating Mean

In statistics, mean is defined as mathematical average of given set of numbers. It is calculated by summing all number divided by total numbers. Statistical formula and representation of mean is given below:

$$\bar{X} = \frac{\sum X}{N}$$

Where:
$\bar{X}$ (X-bar) is the symbol for the mean.
$\sum$ (sigma) is the symbol for summation.
X is the symbol for the scores.
N is the symbol for the number of scores.

Chapter 4

Results

In this chapter, the results of the experiment are presented. To represent the results on more understandable way, they are plotted on different bars and graphs. Showing results in diagram also helps for the comparative analysis of two or more findings. Here, the results are described on two different sections. First section shows the result of white box testing, where low-level performance of the system is determined and second section gives the result of black box testing, where macro performance of the system is determined. All the results represent the performance of first virtual machine (dvm1) on different conditions, such as 1_up, 2_up and so on. Hereafter, N_up represents the state of first virtual machine where N number of virtual machines are running. For example, 10_up means the state of first virtual machine, i.e. dvm1, when it is working with other 9 virtual machines (total 10 VMs are running).

4.1 White Box Testing(Micro Benchmark) Result

In white box testing, Lmbench is used to determine the micro performance of the system where processes latencies, communication latencies, communication bandwidth, file system latency, memory latency and context switching of the system are measured.

4.1.1 Processes Latencies

This is a process creation test where processes are created on three different forms. This test helps to measure the time needed for creating a basic thread of control. The time is measured on microseconds and short time period is

considered better.

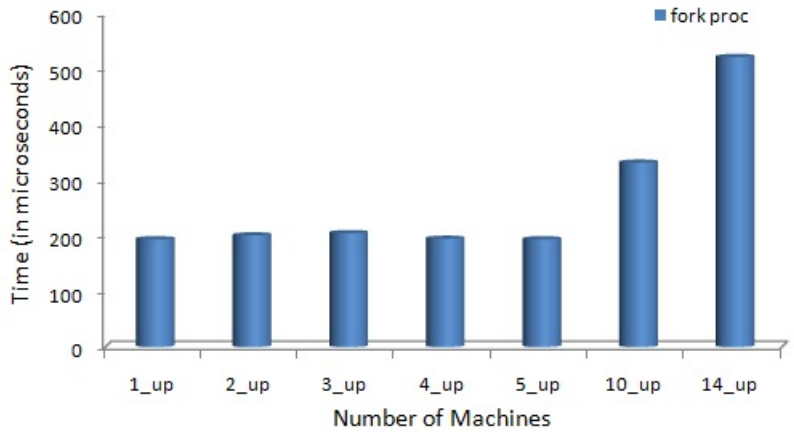

Figure 4.1: fork proc Latency

fork creates a identical copy of calling process which is also called child process. Figure 4.1 shows the latency for forking a new child process and immediately exiting from it. Latency is measured with fork and exit time. The diagram presents forking latency of Debian system (dvm1) in different stages. The latency is almost same around 200 μs upto 5 machines running condition. It is risen when 10 VMs are running, and reach more than 500 μs when 14 VMs are up.

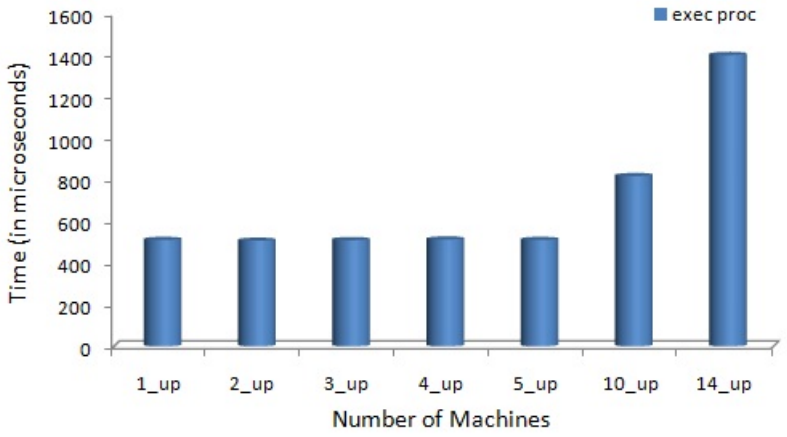

Figure 4.2: exec proc Latency

Measuring exec latency is another way of determining process latency. It is used to measure time, when current process are overwritten with the newly created process and time required to run a new program for that process. In

figure 4.2, latency is 1/3 times higher in up_10 than the up_1 stage. It is measured around 800 μs and 1.4 ms in up_10 and up_14 stages respectively.

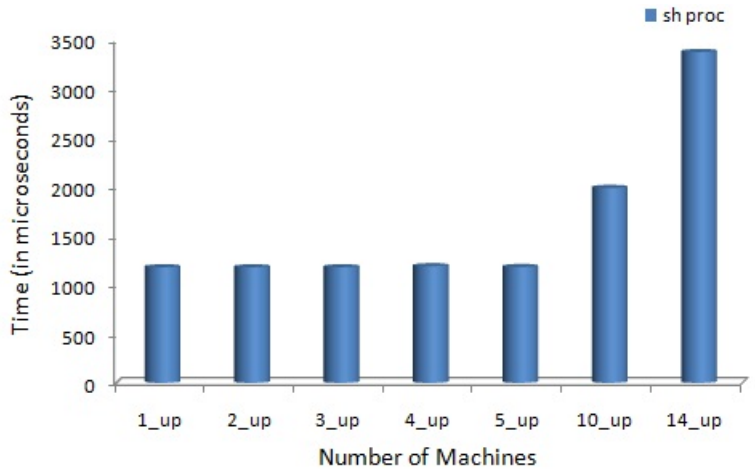

Figure 4.3: sh proc Latency

sh proc latency is shown in figure 4.3. It is a time taken to create a new process, and run a new program by asking system shell for finding and executing that program. sh proc latency is almost 3 times higher in up_14 than up_1 stage.

4.1.2 Communication Latencies

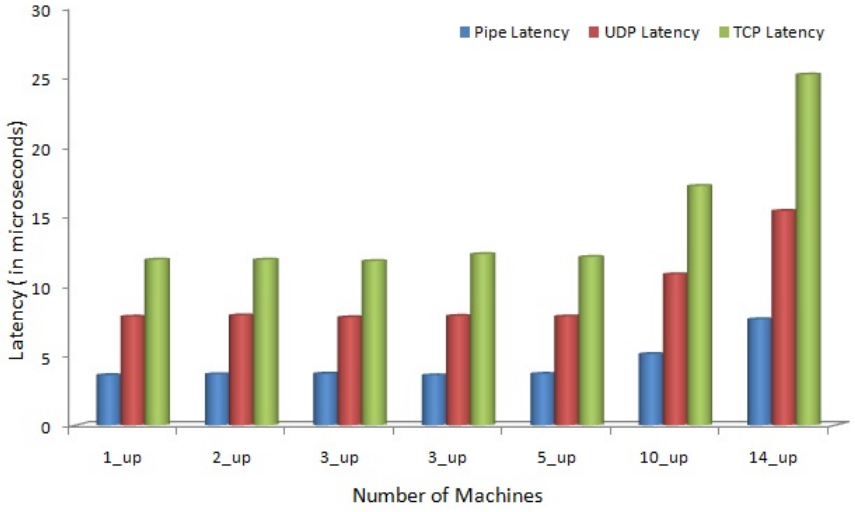

Figure 4.4: Interprocess Communication Latency

Local interprocess communication latency is measured by passing a small sized message back and forth between two processes. Pipe latency is measured by

creating a pair of pipes, forking a child process, and exchanging between them. In TCP/UDP latency measurement, server process waits for connections from client process and message is circulate between them. Figure 4.4 reflects the local communication latencies. All three types of latencies are higher in up_10 and up_14 stages but smaller latency is considered better like in up_1 stage.

4.1.3 Communication Bandwidth

This test is similar to previous section but its main focus is on bandwidth of communication. TCP bandwidth test involves transferring 1 mb data between two processes. Figure 4.5 shows that the bandwidth is gradually decreasing when other virtual machines are started to work on one-by-one. Bandwidth decreases twice on up_14 stage than the stage of beginning, i.e. up_1 stage. It means when system is running alone and with other fewer VMs, it is resulting higher bandwidth. Higher bandwidth is considered better.

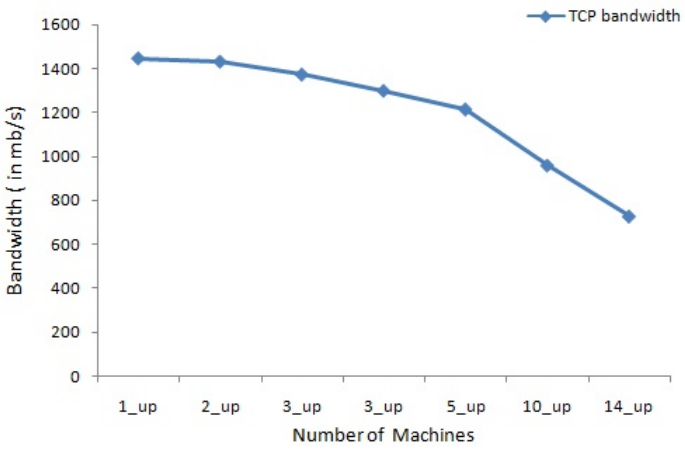

Figure 4.5: Interprocess Communication TCP Bandwidth

4.1.4 Memory Bandwidth

This is a test where memory bandwidth is determined with the help of bcopy() memory copy routine, and results are reported in megabytes moved per second. In this test, source and destination area of memory do not map the same lines hence it measures the actual bandwidth of copy. Figure 4.5 shows the measurement of user-level library bcopy bandwidth for copying data over a varying set of array size on the memory. In large size of data copying (more

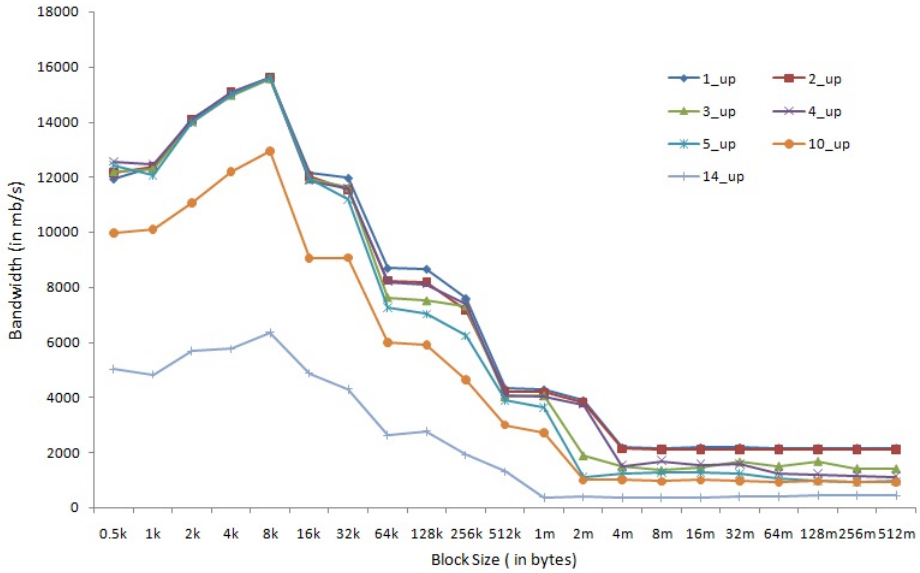

Figure 4.6: Memory Bandwidth using bcopy

than 4 mb), which is in main memory, the bandwidth is almost constant. The bandwidth is better in 1_up stage than others. It is degreased in each consecutive stages, when two or more system are at work. It is measured 2000 mbps in 1_up stage, 1000 mbps in 10_up stage and 500 mbps in 14_up stage.

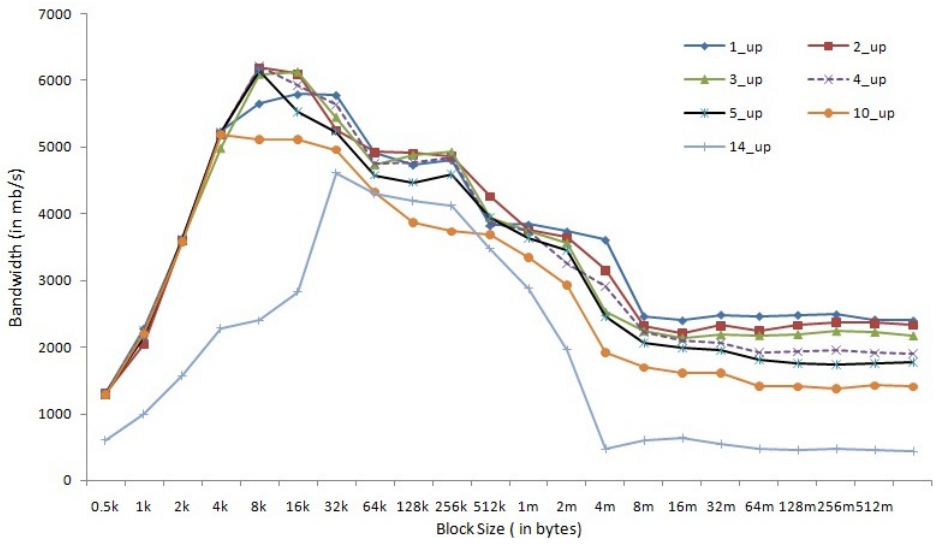

Figure 4.7: Bandwidth of file re-read using read()

File re-read test is similar to above library bcopy() test but in this case, it gives the bandwidth of file re-read operation using read(). reread performance is

usually better than bocpy because it is facilitated to use memory subsystem of the kernel. Figure 4.7 shows the bandwidth of file re-read using read(). The figure clearly shows that the bandwidth is higher in 1_up stage than the bandwidth in all consecutive stages of the system. For example, the bandwidth is around 2500 mbps in 1_up state, but it is slightly greater than 500 mbps in 14_up stage.

4.1.5 File System Latency

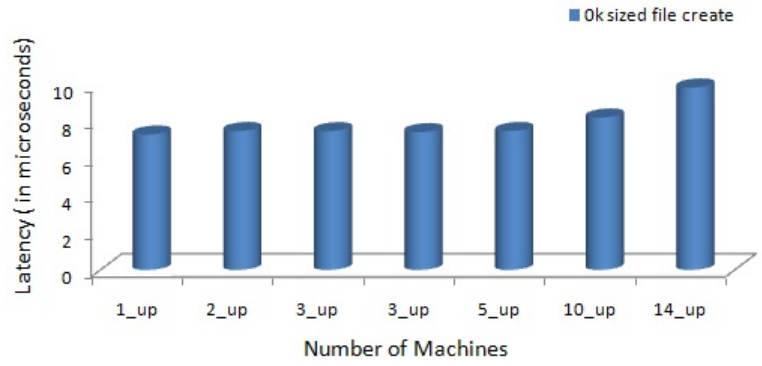

Figure 4.8: File System Latency - 0k file create

In file system latency measurement, lmbench creates and deletes two different sized files (0k and 10k) and measures time of that task. The files are created on same directory and deleted instantly after creation.

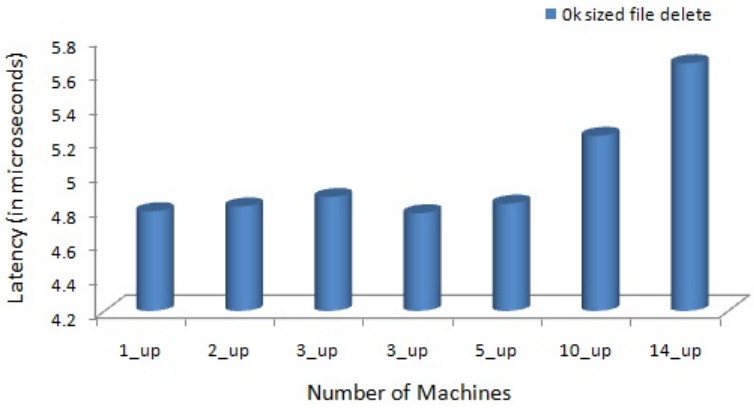

Figure 4.9: File System Latency - 0k file delete

Figures 4.8 and 4.9 shows the file system latency measurement during 0k sized file creation and deletion respectively. The files are very small in size so there is not any significant latency difference for different N_up conditions of the system. However, small growth can be seen when large number of virtual machines are in up state.

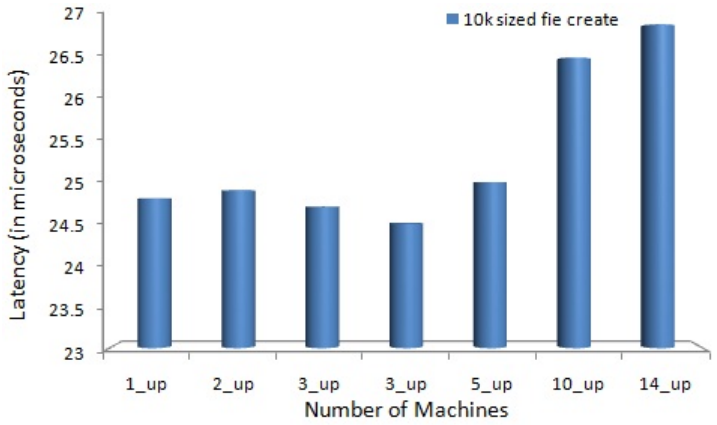

Figure 4.10: File System Latency - 10k file create

Similarly, figures 4.10 and 4.11 represents the file system latency measurement during 10k sized file creation and deletion respectively. It is seen that there is not that much difference in latency during few number of virtual machines are in up states. Latency is increased when large number of virtual machines are working on the same time. Systems having small file system latency are considered as better systems.

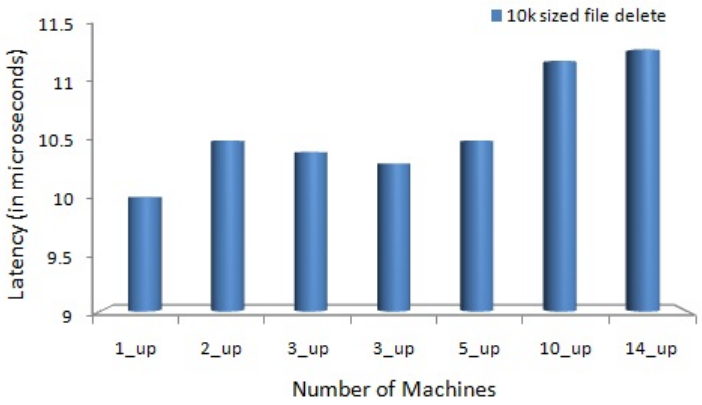

Figure 4.11: File System Latency - 10k file delete

4.1.6 Memory Read Latency

This test is for determining memory read latency for various memory sizes and strides like 16k, 32k, 64k, 128k etc. The results are reported in nanoseconds per load. The benchmark test varies in two parameters, array stride and size. Only latency in stride of 128k is presented in this report.

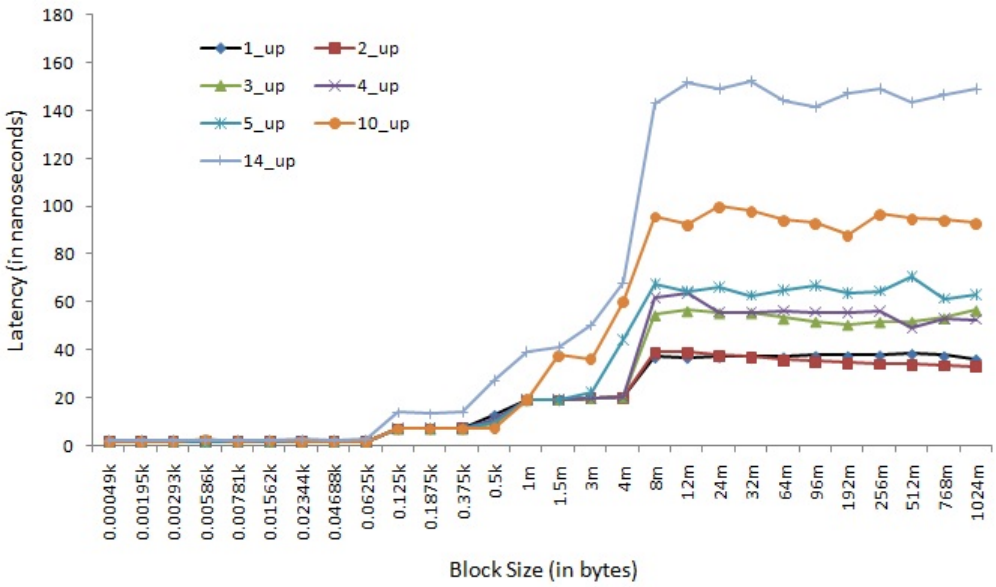

Figure 4.12: Memory Read Latency at 128k Stride

Figure 4.12 clearly shows onboard cache latency (from array size of .00195k), external cache latency (from array size of .0125k) and main memory latency (from array size of 8m). When virtual machines are started to work, memory read latency is increased also. These latencies are several times higher in up_10 and up_14 stages.

4.1.7 Context Switching

Context switching is a time needed for the process of saving state current process and loading the state of next process. Lmbench determines the latency of context switch with number of processes (2, 4, 8, 16, 24, 32, 64, 96) and size (0K, 4K, 8K, 16K, 32K, 64K). Only context switch latency of 64P of 4K ,64P of 32K and 2P of 0K are chosen in this research.

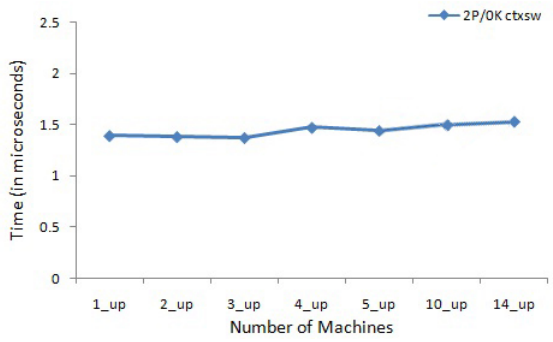

Figure 4.13: Context Switching: Total Processes 2, Size 0K

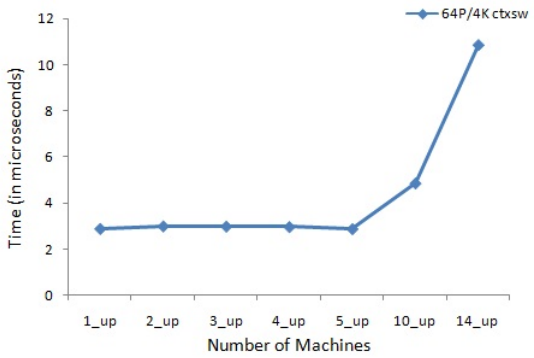

Figure 4.14: Context Switching: Total Processes 64, Size 4K

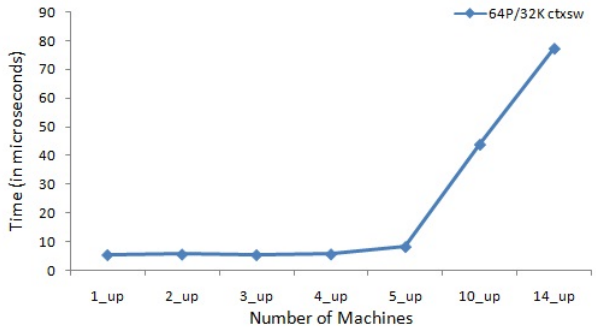

Figure 4.15: Context Switching: Total Processes 64, Size 32K

Figures 4.13, 4.14 and 4.15 show the context switching latency of process in dvm1. Lower latency is considered as better. In the figure 4.13, there is not that much difference on latency, as this result is for 0K sized processes. But in figures 4.14 and 4.15, a huge rise is seen when large number of virtual machines are up. However until 5_up stage, context switching is not hugely affected by the scalability. It is may be the powerful host machine. The latency is risen high in 10_up and 14_up stages.

4.2 Black Box Testing(Macro Benchmark) Result

In black box testing, macro performance of system is determined. Macro performance means overall performance of a entire system or subsystem. In this research, overall system performance is determined with the help of file subsystem, processor subsystem and network subsystem testing. The result of memory read / write bandwidth obtained from lmbench benchmark is also taken to reflect the memory virtualization overhead.

4.2.1 File Subsystem Benchmarking

The result of file subsystem benchmarking by bonnie++ is shown in following figures. Generally, bonnie++ performs series of tests on file system and presents the results in two sections. They are file I/O test and file creation test. From the massive result of bonnie++, only the first part is presented in this paper. They are sequential output, sequential input and random seeks. In this test, 5G file is assigned to bonnie++ to do the testing.

Sequential output per block and rewrite is shown in 4.16. In sequential output per block test, bonnie++ creates files using write(). When writing the 5G file using file efficient block writes, bonnie++ gives the result under this filed. The result is in k/sec. Similarly in sequential output rewrite test, bonnie++ generates result in k/sec for creating, changing, and rewriting each block on the given 5G sized file.

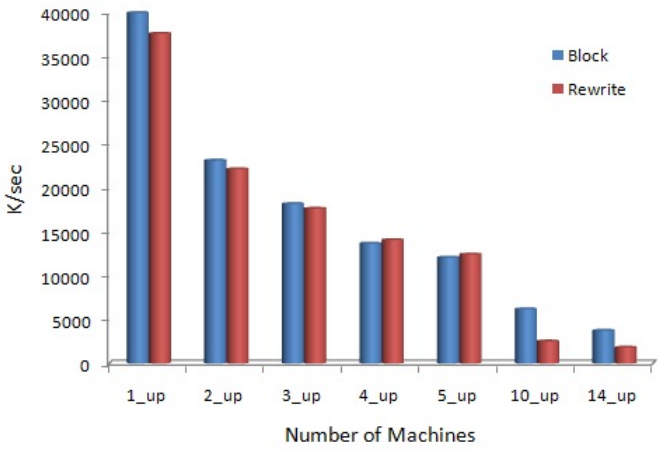

Figure 4.16: Bonnie++: Sequential Output

In figure 4.16, it is clearly seen that the scalability has made huge impact on the disk performance of the system. In both tests of sequential output i.e. block write and write, processing speed is more than 36 mbps when system is working alone. As the number of machine increased, the processing speed is degreased in each next stage of system. In 14_up state of the system, the result of both the tests are quite low, i.e. less than 5 mbps.

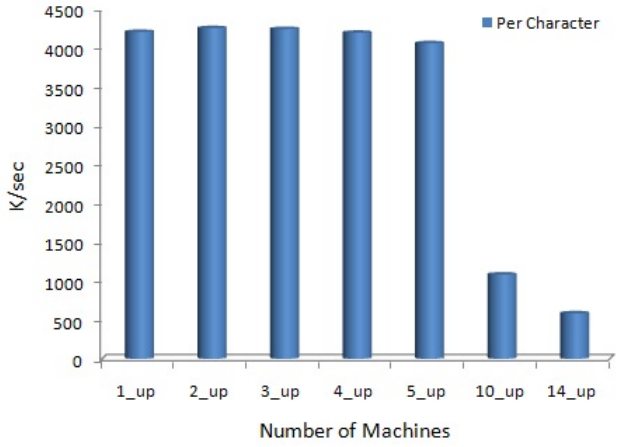

Figure 4.17: Bonnie++: Sequential Input

Figure 4.17 show sequential input per character test. In this test, bonnie++ reports an input rate while file is read with millions of getc() macro invocations. The figure is slightly odd in nature because the rate is almost same upto 5_up state but has decreased huge like previous result, almost 4 times less, when

large number of machines are working together.

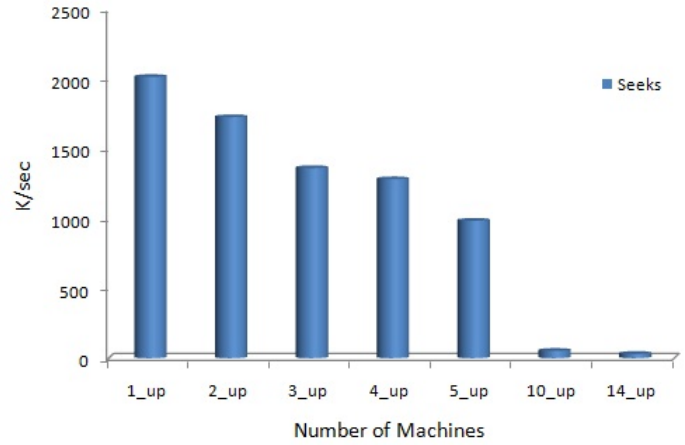

Figure 4.18: Bonnie++: Random Seeks

Random seek test is the best way to know how fast array can find data. In this test, bonnie++ created 4 child processes, and had them execute to thousands of seeks to random locations in the file. Figure 4.18 shows random seek test. The figure clearly shows that it is highly affected when the large number of machines are working together.

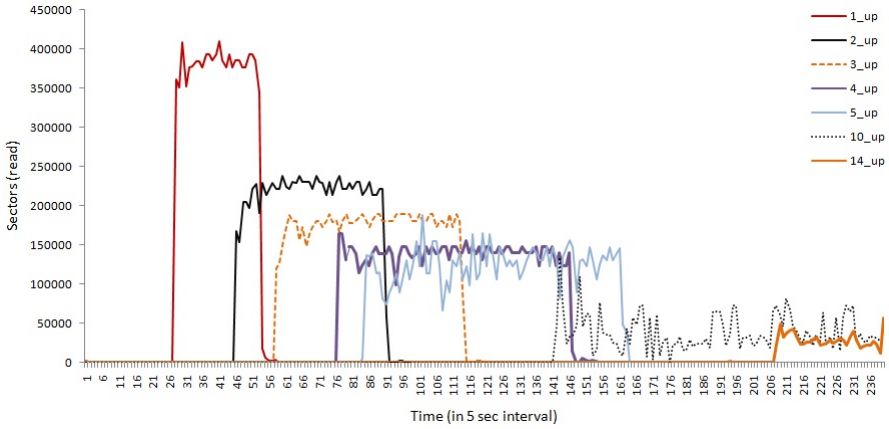

Figure 4.19: Bonnie++: sectors read

As mentioned in previous sections, during execution of bonnie++ the result of sectors read and sectors write is recorded from */proc/diskstats/* directory by running separate perl script. This result is shown in figures 4.19 and 4.20 re-

spectively. In both figures, y-axis represents a total number of sectors read or write in each 5 seconds interval of time.

During sectors read, system on 1_up state is performing well with reading large numbers of sectors in short period of time. As the number of virtual machines increased, the performance is degraded in each next stage of system. The numbers of sectors read are decreased and it also takes long time to complete. Not only these but also takes long time to begin the read operation. The graph clearly shows that both in up_10 and up_14 states, read operation takes more than 10 minutes to begin, very less number of sectors are read, and taking long time to complete (graph is not shown after 20 minutes).

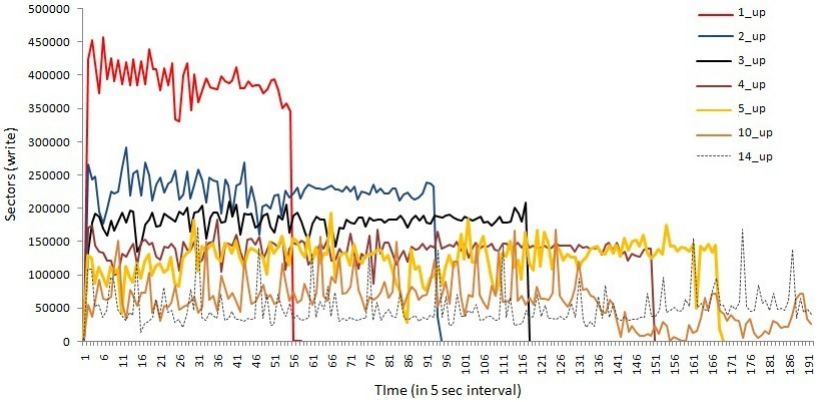

Figure 4.20: Bonnie++: sectors write

When bonnie++ starts to run, it starts its test by writing operation which can be seen in figure 4.20 also. Like read operation, large numbers of sectors are written in each 5 seconds interval of time when system is in 1_up state. As more virtual machines are started to work, the total number of sectors write are decreased in each consecutive stage of system, and the time to complete the same task (writing) in 1_up state is taking very long time in other N_up stages. For example, it is 4.5 and 8 minutes in 1_up and 2_up stages respectively.

4.2.2 Memory Virtualization Overhead

To understand the overhead of memory virtualization, the result of lmbench is presented in this section. Memory virtualization overhead is determined with the help of bandwidth. In this test, Lmbench uses bw_mem_rd and bw_mem_wr

functions to calculate the overhead of memory read and write operations respectively. In both operations, lmbench specifies amount of memory, zeros it, and then times the reading/writing of that memory as a series of 4 byte integer stores and increments.

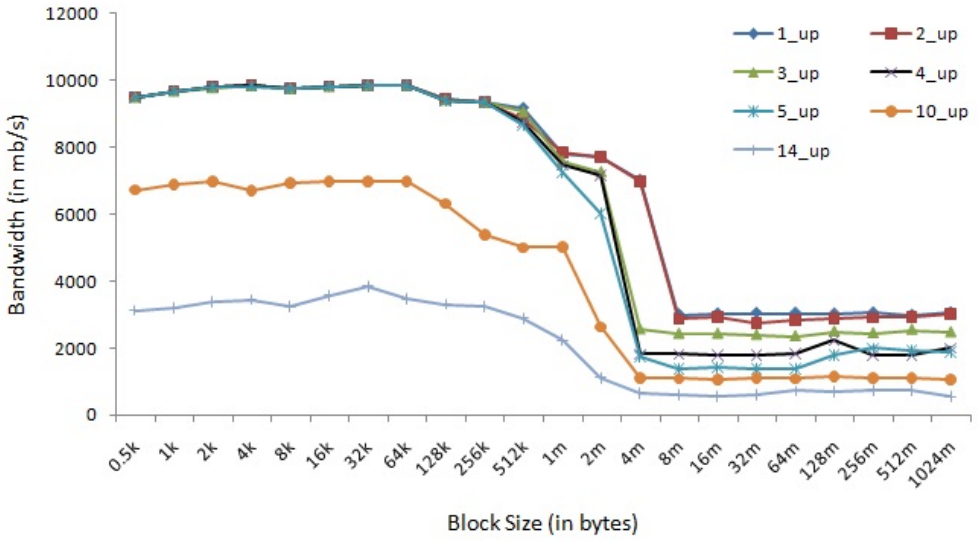

Figure 4.21: Bandwidth of Memory Read Operation

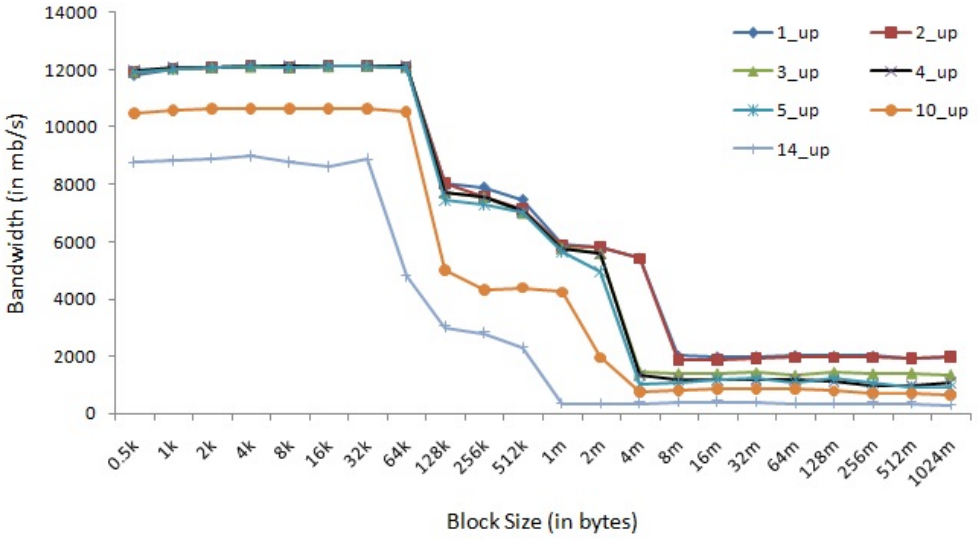

Figure 4.22: Bandwidth of Memory Write Operation

Figure 4.21 and 4.22 shows result of memory read and write bandwidth respectively. In both cases, when the number of virtual machines increase, the bandwidth is decreased. In the beginning of both figures, the bandwidth of

read/write operation for level 1 and 2 cache memory can be seen. The read-/write operation on main memory starts only from the block size of 8 mb. In figure 4.21, the bandwidth of memory read operation is around 3000 mbps in the 1_up state of system, and decreased almost half when system is in 5_up stage. It is reached around 500 mbps when the system is in 14_up stage. Similar result can be seen in figure 4.22 for the bandwidth measurement of memory write operation. The bandwidth is decreased almost four times when the system is in 14_up stage.

4.2.3 Network Throughput Benchmarking

The bandwidth of network is determined with the help of NetIO benchmark. Figure 4.23 shows the network bandwidth of receiving tcp packets from the clinet. Similarly, figure 4.24 shows the network bandwidth of sending tcp packets back to the client.

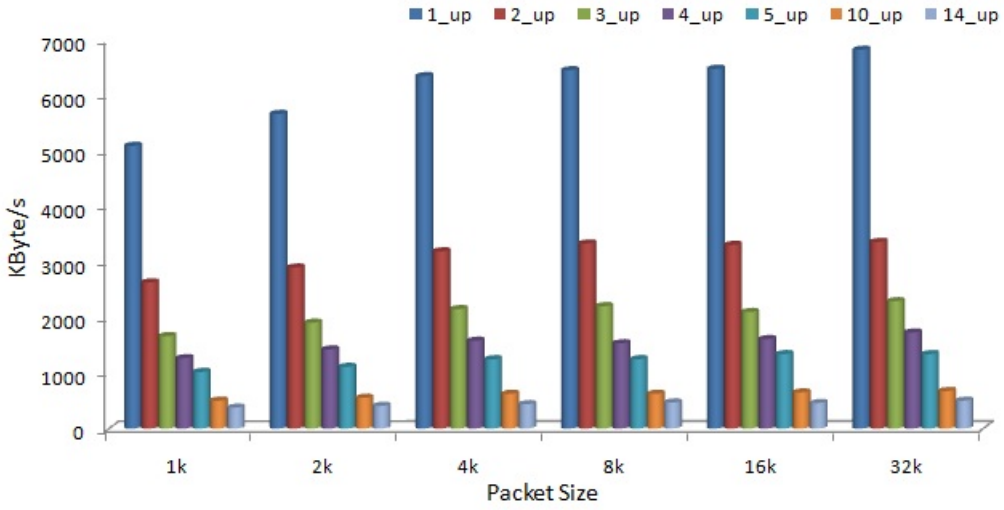

Figure 4.23: Network Bandwidth using TCP protocol - receive rate

In both figures, the bandwidth is shown on different packet size of 1k, 2k, 4k, 8k, 46k and 32kb. The receiving rate of all packet size is high in the system of 1_up state. As the number of virtual machine increased, the packet receiving rate is started to fall down. In 1_up state of figure 4.23, the receiving rate is more than 5 mbps in all conditions of different packet size. In 2_up state, it is decrease almost 50 %. The bandwidth is more affected in 14_up state. It is

less than 500 kbps in that stage. In this test local VMs are made servers and hp3.vlab.iu.hio.no (a remote machine, ip 128.39.73.234) is made client.

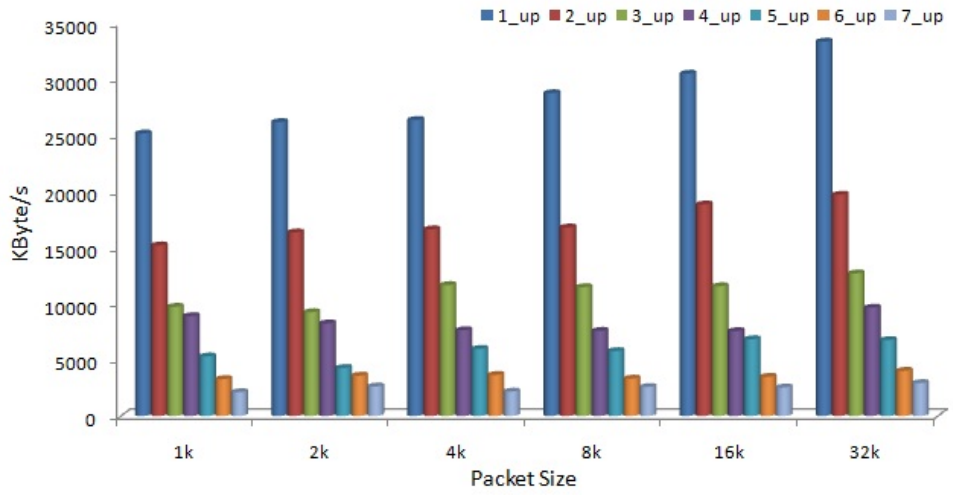

Figure 4.24: Network Bandwidth using TCP protocol - sending rate

The figure 4.24 shows the sending rate of packet from the server to the client. The pattern of graph is very similar to previous result except its transmitting rate. The sending rate is more than 25 mbps in all size of packets in 1_up state and decreased gradually when N (N = 2 or more) machines are started to work.

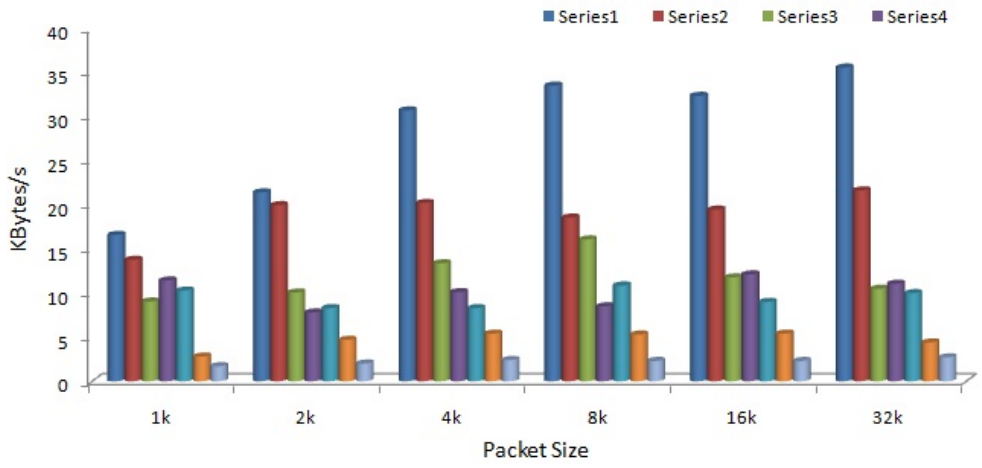

Figure 4.25: Network Bandwidth using UDP protocol - receive rate

Figure 4.25 is the result of bandwidth measurement for udp packets. This result is little bit inconsistent than previous two results. However, an impact

can be seen on the network performance for all size of packets when two or more machines are turned-on. Specially, it is quite low in 10_up and 14_up states. In this test local VMs are made clients and s160487@studssh.iu.hio.no (a remote machine, ip 128.39.74.65) is made server.

4.2.4 Processor Virtualization Benchmarking

The system information obtained from *mpstat* Linux command is shown in following figures. The output of *mpstat* command looks like as follow:

```
01:43:42 AM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %
    guest   %idle
01:43:42 AM  all    3.81    0.00    0.02    0.02    0.00    0.00    0.00
       0.00   96.16
```

From this output, only usr, sys and idle are chosen and result is shown in figures 4.26, 4.27 and 4.28 respectively. In these figures, the CPU utilization is shown in percentage. These results represent the system information during the execution of CPU stress generator script (load.pl) for 10 minutes.

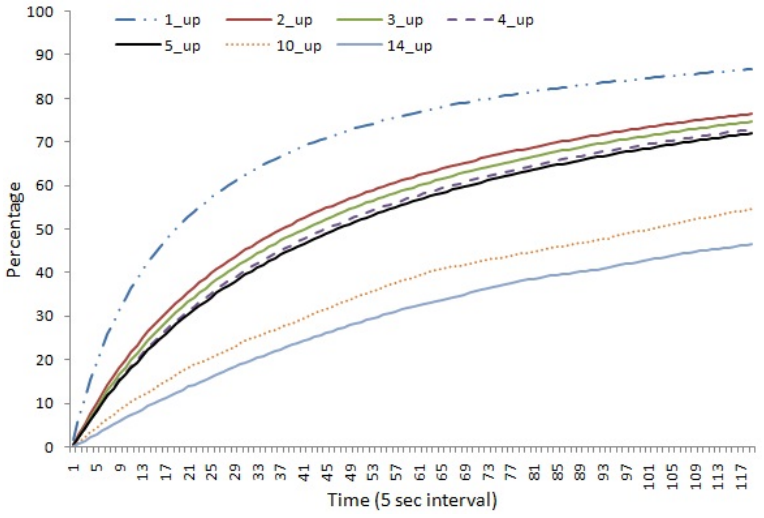

Figure 4.26: Percentage of CPU used by user related processes

The figure 4.26 shows the percentage of CPU used by user related processes (user level application). The CPU usage is reached upto 85% after 10 minutes of script execution in 1_up stage, but only 40% of CPU is used for the same task when system is in 14_up state.

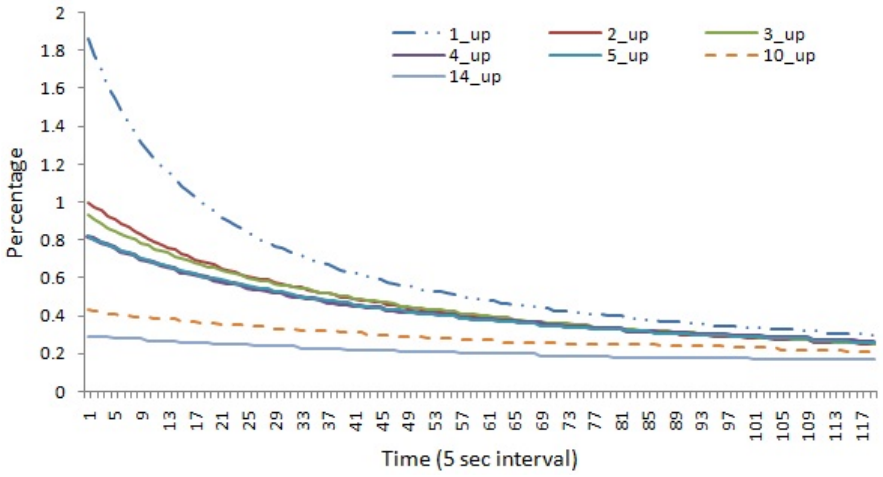

Figure 4.27: Percentage of CPU used by system related processes

The figure 4.27 shows the percentage of CPU used by system related process (kernel level application). The CPU use is high in 1_up state but when other VMs starts to work, the CPU utilization is decreased. Similarly, figure 4.28 shows the percentage of CPU used by idle processes. It also has a clear scalability impact. When multiple VMs start to work, the CPU becomes busy and as seen in figure, the percentage of CPU usage by idle processes increases. When the system is in 1_up stage, the CPU usage is 20% after the 10 min of script run. It is reached to 70% when the system is in 14_up stage.

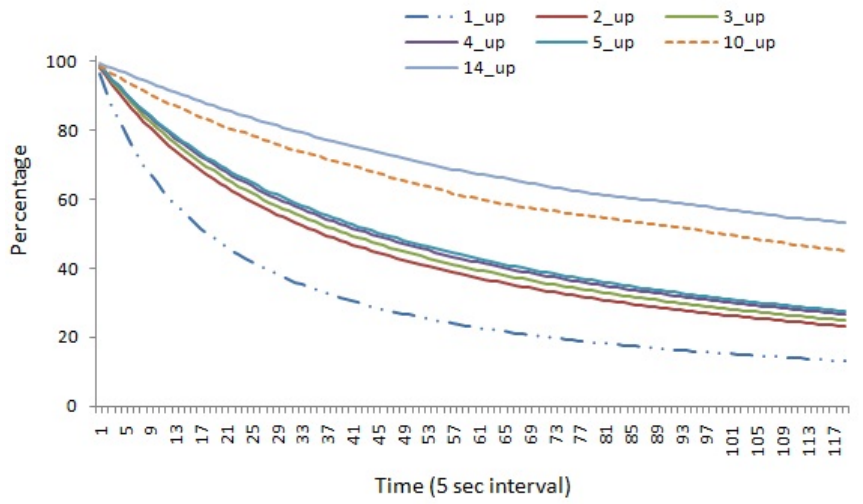

Figure 4.28: Percentage of CPU used by idle processes

Chapter 5

Discussion

5.1 Review of Methodology

Before starting the analysis of the results, it would be better to recall the problem statement of this research and the approaches taken for it. The core interest of this research is to find out the performance of virtual machine on the presence of other virtual machines under Kernel-based Virtual Machine environment. Particularly in this research, literature survey helped to understand the related works very closely and showed the best way to conduct this research in a more formal and correct way. The time of literature survey is the most crucial time for all researchers because a minor mistake may ruin the whole experiment. To address the stated problem, two main approaches were considered and this whole research was carried out on the basis of those approaches. In the first approach, the system was considered as white-box for analysing its micro-performance by determining bandwidths and latencies of system operations, such as process latencies, IPC latencies, IPC bandwidth, context switching etc. In the next phase, the system was considered as black-box for analysing its macro-performance on the memory virtualization, processor virtualization, disk virtualization and network virtualization.

As the main focus of this research is to understand the scalability impact on the performance on virtual machine, Debian GNU/Linux was chosen as a guest operating system. The only reason for choosing Debian as a guest OS is because it is one of the most popular Linux operating systems. The reason of selecting KVM hypervisor is also due to its growing popularity. This is quite young hypervisor on the market, but it has been able to leave a positive im-

pression among computer users within this short period of time.

Another important task is to secure the whole experiment. The experiment was conducted on a machine having public IP, so that it could be accessible directly from anywhere. A small firewall rule was created and run on the host machine. This rule only allows the users of certain networks of the university to access the host machine remotely. It was not the finest solution, however it completely reduced the problem of malicious activity from the outside of the university network.

The experiment was started by testing the system performance by using different benchmarking tools. The tools were chosen on the basis of their easiness for installation, configuration, output and documentation available in the Internet. Lmbench benchmark was chosen for the micro-performance of the system for analysing the bandwidth and latency of data movement between memory, cache, disk and network. As the macro benchmarking reflects the overall system performance, macro performance of the system was analysed on three different fields. They are memory virtualization, processor virtualization, network virtualization and disk virtualization. NetIO benchmark was chosen for network bandwidth testing for TCP/UDP packets and bonnie++ benchmark was chosen for testing disk performance. Once again, the memory virtualization overhead was analysed with the help of Lmbench tool. In each phase of testing, the same hardware was used, which creates a fundamental background to evaluate the system performance in different stages.

5.2 Analysis of Result

Micro-Performance Analysis

The experiment was started by using Lmbench tool for micro-benchmarking test of the system. The guest operating system on all virtual machines was Debian, so it was necessary to analyse the performance of the system in low-level kernel primitives. Kernel is a core part of a operating system, which acts as a communication medium between user level applications and data processing units. As this experiment was conducted on virtual environment, all those data processing units of each VMs were virtual also. So, it was important

to know how does the sharing of same processing unit among multiple machines affect the vitrual machine's performance, how is the internal working of system operations, such as processes latencies, communication latencies, communication bandwidth, file system latency, context switching etc. These lower level system operations are very critical to the high-level applications also.

A system call is a request from the user's program to the kernel to get some sort of services from the kernel. For example, if we need to create a program to write on file, we have to do system call to request operating system to write on that file. Generally when system call is made, the kernel has full control on program execution and arguments until the call finishes. Hence, system calls have a vital role between system kernel and user level system applications. There are several types of system call but generally, they can be separated on 5 major groups. They are:

- Information Maintenance:
  - get/set time or date
  - get/set system data
- Process Control:
  - load, execute
  - create / terminate process
- Device Management:
  - request / release device
  - get/set device attributes
- Communication:
  - create / delete communication connection
  - send, receive messages
- File Management:
  - create / delete file
  - open, close

In this experiment, all the systems (VMs) were virtual. Sometime they run alone and sometimes with other VMs. As all the systems running on virtual

environment share the same physical resource, the result of the system operations were found quite interesting. The performance of system is highly affected by the presence of other virtual machines.

Process management is related to processes and responsible for their creation, deletion, execution and switching. To understand these operations process latencies were measured with respect to time. fork() causes the system to create new child process with new ID. fork()+execve() create a new process and run new program. fork()+/bin/sh creates a new process and run a new program by asking system shell for finding and executing that program. In all cases, the lower latency is considered better because lower latency means that the system operations are quick. The result shows that the latencies were lower in those states when VM was running alone and running with other fewer VMs. As the number of VMs increased more, latency was measured higher.

Interprocess communication is a process of exchanging data between processes. In Linux, local IPC mechanism can be of different types, such as pipes, sockets, shared memory, message queues etc. In this experiment, latency and bandwidth of TCP/UDP sockets and pipes were measured only. In the case of local IPC latency measurement, latency was found lower for all types of IPC mechanism (tcp, udp and pipe) when VM was running alone. It was not highly affected even working along with other fewer number of VMs but if larger number machines work together, it was affected more. Lower latency informs that the the data flow between server and client is fast. The results also shows that the TCP had higher latency than UDP and pipe. It is because UDP does not have any type of flow and error control and thus performs faster than TCP. Pipe performs even better because it is much simpler IPC mechanism having unidirectional data flow. In the next experiment, local communication bandwidth was measured for TCP communication. Bandwidth refers how fast data are transferred over a channel so high bandwidth is considered as better always. The result shows that TCP communication bandwidth of a system was also highly affected by the presence of other VMs.

A basic concept for determining the performance of computer system is the movement of data on it. The main concern of memory bandwidth measurement in this experiment was to find out the rate at which data can be read

from memory by the processor and the rate at which data can be copy from one location to another location of memory. During copy of data, the source and destination area of memory do not map the same lines hence it measures actual bandwidth of copying. In this experiment, the result shows that the actual memory bandwidth measurement started from the array size of 8MB. In the beginning, the graph shows high bandwidth because Lmbench measured cache memory before main memory, but this experiment does not concern on cache memory. Similarly in the measurement of bandwidth for re-read, the bandwidth was measured via read operation. re-read performance is usually better than bocpy because it is facilitated to use memory subsystem of the kernel. In both measurements, the bandwidth was high in the single machine operation but measured low when multiple numbers of VMs started to work together.

In file system latency measurement, the time was determined for the creation and deletion of files of different size. The Linux system uses modern ext3 file system which is fast in performance. Generally, creating task takes longer time than deleting, which also can be seen in the result of file system latency measurement. The result shows that the latency of creating and deleting files was higher in the case of 10k file size than 0k file size. It is because the system need less time for those tasks for small size files. Similar as the above results, the file system performance of the VM is also affected by the presence of other VMs.

Context switching measurement is an important test of micro-benchmarking. It is defined as a time needed for suspending progression of one process and resuming execution of other process that had been suspended previously. Small context switching time represents the better processing of process or threads in the kernel level which is always desired. In this experiment, the context switching of same number of process in different states of VMs (1_up, 2_up and so on) were risen high and risen more when VM was running in up_10 or more states, but due to the size of process for 0k size even with its large number, there was not significant difference in context switching in all states of the system. It was also measured that in all N_up states of the system, as the number of process increased, the latency was risen high. It is because when the number of process increases, the load on CPU also increases which creates delay on processing.

Macro-Performance Analysis

Macro performance reflects the overall performance of a entire system or subsystem. In this research, the system was considered as a black box where macro-performance on the processor virtualization, memory virtualization, disk virtualization and network virtualization were measured. Generally in black-box testing, the internal mechanism of system is discarded.

The aim of CPU virtualization benchmarking was to determine the performance of virtual CPU of virtual machine and its performance on the presence of other vCPUs. Generally when user level application are running, they create a massive workload on CPU and system may slow down. To understand these things, a simple CPU workload generator was run in this experiment on different states of the system. The result shows that the CPU used by user related process were increased smoothly on each states of system but interestingly even with smooth increase, the level of percentage of CPU usage was decreased on each N_up states of system. Similarly, the level of CPU usage percentage by user related process were measured less when multiple VMs started to work. The percentage of CPU usage by idle processes also has a clear scalability impact. When multiple VMs start to work, the CPU becomes busy and as seen in the figure, the percentage of CPU usage by idle processes increases.

The main concern of memory virtualization benchmarking was to find out the rate at which data can be read and write to and from the memory by the processor. In this experiment, the result shows that the actual memory bandwidth measurement started from the array size of 8MB. In the beginning, the graph shows high bandwidth because of the reading of cache memory before main memory. However, the bandwidth of main memory shows that both read and write operation of the system were affected by the presence of neighbour VMs. In each next stage of VM, the bandwidth is decreased than the previous stage of VM. The result also shows that the bandwidth of read operation was always better than the bandwidth of write operation in all states of system which is universal truth in computing.

To understand the virtualization of disk, Bonnie++ was used. KVM uses vir-

tual disk I/O with modified QEMU process. When disk I/O requests come from virtual machine, they are trapped into the kernel mode and the kernel schedules QEMU to simulate the disk I/O operation. This process goes complex when two or more virtual machines are running and requesting I/O operation in the same time. Exactly same scenario was created during the experiment for disk performance testing. The result shows that both sequential input and sequential output were measured low when the system was working with one or more VMs. Similarly, three major scalability impact could be seen on the sectors read operations. First, when system worked alone, the total sectors read was quite high. Second, the read operation was started very fast and third, the whole read operation was completed on short period of time. When other VMs were started to work, all the given operations were affected, and the system gave low performance in each consecutive stage. Similar type of performance impact can be seen on write operation. Large number of sectors were written in 1_up stage of the system. The operation was completed quite fast also. As several VMs started to work, sectors write operation was affected in each next stage of system.

Network virtualization benchmarking was done to find the speed of transmitting and receiving data packets to and from the virtual machines. Network performance of virtual machines depends on the channels and the interfaces through which data is exchanged. In KVM, the packet coming from VMs are only sent via network interface to outside after being interrupted into user-space by the Linux kernel, which may affect the network performance. When two or more number of VMs start to run, they have to share same communication channel which leads the network performance loss. This can be seen in the network testing of this experiment also. The result shows that as the multiple number of machines started to work, the network performance (both receiving and sending rate) of the system in each consecutive stages were decreased. The receiving rate of UDP packets were little bit inconsistent than TCP, because of the network congestion and UDP is unreliable protocol also. However, a clear impact of scalability can be seen on the network performance in both protocols.

5.3 Difficulties

Some problems and difficulties were encountered during the course of the experiment. They were not complex, but were very time consuming. Some problems were fetched during the set-up of virtual environment and some were found during system testing.

One irritating issue was a firewall problem, but it was not due to the firewall of host machine. The university has its own firewall on the gateway to control the traffic to and from the university network. The firewall rule black-listed the IP of host machine frequently. It reported the error due to multiple ssh to outside in short time. The host machine was secured with strong password and multiple ssh to outside network is not required for this experiment. So, the black-listing was quite strange. Usually it happened in the period of network testing. During bandwidth testing, packet of different size were sent between clients and servers. This was a only work which was related to traffic (packets). Still, the problem is unknown.

Another problem is also related to the network bandwidth analysis for UDP packets. The default policy of firewall on the host machine drops all the incoming traffic. So, some couples of rules were added for port forwarding for TCP/UDP packets coming from clients to local VMs (servers). It worked for TCP packets but did not work for UDP. The NetIO benchmark documentation have clearly mentioned that NetIO works for both TCP and UDP packets. Finally, network bandwidth analysis for UDP packets was done by making local VMs as clients. For this case, no firewall rule was necessary because default policy of firewall for outgoing traffic was accepted.

Some minors but time consuming problems were encountered during implementation of benchmarking tools. As mentioned before, one important reason of choosing the benchmarking tools was their better documentation than other similar tools. During experiment, it was noticed that some minor steps were not mentioned neither in documentation nor in README file of the respective tools. For example, making some files executable, installing compiler, location of result etc. Sometime these issues took a whole day to figure out.

5.4 Future Work

Although this experiment is done on KVM virtual environment with large number of virtual machines having Debian OS, some additional works can be done to understand the system more closely. In this experiment, the virtual machines were supposed as systems without any service. Generally, virtualization technique is implemented to run multiple isolated virtual machines as servers. The parties may implement the VMs for different purpose, for example web servers, database servers, mail servers, game servers etc. On the top of this research, additional benchmarking can be done for specific servers for their services. Such as: analysing the performance of database server when it runs with other multiple servers and similar test for other service providers. Further more different operating systems can be chosen for the experiment also. This will help for decision making of which is the best operating system for certain service.

Sometime own interest and sometime the necessity motivates researchers to do experiment. As this research is done under KVM environment, future work can be done on different virtualization platform, such as Xen, Vmware, openVZ etc. The research on these hypervisors will be very helpful in decision making for the system administrator to choose the best platform that have less scalability impact. Another interesting work can be done during the live migration of virtual machines. Generally, live migration allows virtual machines to move between different physical machines. If live migration is possible then it is important to know that how the system performs during migration.

Chapter 6

Conclusion

With numerous features such as live migration, dynamic load balancing, server isolation, consolidation and so on, virtualization technology is in the peak of popularity. The most common and simple way to define server virtulaization is that it allows to create a large number of fully functioning servers(computers), also called virtual machines, on the top of underlying physical hardware. Now a striking question, how many virtual machines per system?, is always moving around and answer comes in mind without any hesitation. Specially in this scenario if decisions are made on the basis of general discussion or own perception, it may be harmful to the whole virtual environment. This research was done to address the question mentioned before so that system administrator can predict the impact of system performance due to the scalability before choosing the number of machines for his/her own virtual environment.

The result of this research shows that the scalability has huge impact on the performance of system. Either in micro-performance test or in macro- performance test, the system performance has decreased when multiple number of machines work together. In micro-performance testing, the result shows that the system is performing slightly lower in each consecutive stage upto 5_up state. When system runs along with 9 VMs, the performance is decreased and decreased even more when more VMs are added. Alike micro-performance, the macro-performance result also shows a sharp performance degradation on virtual machine when one or more VMs starts to work together.

In server virtualization, each virtual machines works as a isolated physical machine having their own CPU, memory, disk, network interface, but in real-

ity these components are not real. They are virtual in nature and are provided by hypervisor to the respective virtual machines. Hence, activities of reading, writing, executing or other tasks on one virtual machine means these activities are doing on same physical(host) machine. For example, in this experiment, bonnie++ reads, writes, deletes files on the disk of respective virtual machine but in that time, in reality, it is doing those tasks on the hard disk of the host machine. If there is another virtual machine running, it means it is also using the disk of same host machine. Hence, execution of bonnie++ on first VM affects the performance of the other VM and vice-versa. Similarly, if two or more virtual machines wants to communicate outside of the host machine, they have to share the same network interface provided by the hypervisor, and there will be a huge performance penalty in the network bandwidth. In this research, both micro and macro performance test show similar type of result of performance penalty. The research has found a clear performance degradation of virtual machine when it runs with two or more number of virtual machines.

A noteworthy fact is that this experiment was done on the host machine having pre-determined system specification mentioned on section 3.4 so that the result is not a unique representation of all other similar systems. Even if one more VM is added on this experiment, the result may be different. Similarly if other guest OS is chosen instead of Debian, the performance may be different than Debian. On the other hand, choosing the number of VMs totally depends on the hardware configuration of the host machine. There is no rule to specify the number of VMs per system. It entirely depends on how much powerful the host machine is, and how powerful VMs does the user want to create. More importantly, this research has clearly figured out a pattern of performance degradation of virtual machines when it runs along with other VMs. The report will be immensely helpful for decision making before implementing same or similar environment.

Bibliography

[1] Todd Deshane, Demetrios Dimatos, Gary Hamilton, Madhujith Hapuarachchi, Wenjin Hu, Michael McCabe, Jeanna Neefe Matthews. Performance Isolation of a Misbehaving Virtual Machine with Xen, VMware and Solaris Containers. *Clarkson University*.

[2] Christopher Clark, Keir Fraser, Steven Hand, Christian Limpach, Ian Pratt, Andrew Warield. Live Migration of Virtual Machines, *University of Cambridge Computer Laboratory* 15 J Thomson Avenue, Cambridge.

[3] Andrew Whitaker, Richard S. Cox, Marianne Shaw, Steven D. Gribble. Rethinking the Design of Virtual Machine Monitors, *niversity of Washington*, USA.

[4] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast Transparent Migration for Virtual Machines, *VMware, Inc*.

[5] G. W. Dunlap, S. T. King, Peter M. Chen. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. *Department of Electrical Engineering and Computer Science*, University of Michigan, USA.

[6] Jianhua Che, Congcong Shi, Yong Yu, Weimin Lin. A Synthetical Performance Evaluation of OpenVZ, Xen and KVM. *IEEE Asia-Pacific Services Computing Conference*.Pages: 587-594, China, 2010.

[7] Prakash H R, Anala M.R, Dr.Shobha G. Performance Analysis of Transport Protocol During Live Migration of Virtual Machine. *Indian Journal of Computer Science and Engineering*.

[8] Carl J. Young. Extended architecture and hypervisor performance. *In Proceedings of the workshop on virtual computer systems*, pages 177-183, New York, NY, USA, 1973.

[9] Dawei Huang, Jianhua Che, Qinming He, Qinghua Gao. Performance Measuring and Comparing of Virtual Machine Monitors, *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*,Pages: 381-386, 2008.

[10] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS monitoring and performance profiling tool. *Technical Report HPL-2005-187*, 2005, HP Laboratories, Palo Alto, CA, USA.

[11] Aravind Menon1, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, Willy Zwaenepoel. Diagnosing Performance Overheads in the Xen Virtual Machine Environment. *Internet Systems and Storage Laboratory*, HP Laboratories.

[12] Moses Mungai. Performance Analysis of Different POSIX Operating Systems as Virtual Machines. *Master thesis - University of Oslo*.

[13] Aaron B. Brown, Margo I. Seltzer. Operating System Benchmarking in the Wake of Lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture, *Harvard University*.

[14] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauery, Ian Pratt, Andrew Warfield. Xen and the Art of Virtualization. *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles(SOSP)*, Page: 164-177, 2003.

[15] J. Bradley Chen, Yasuhiro Endo, Kee Chan, David Mazieres, Antonio Dias, Margo Seltzer, and Michael D. Smith. The Measured Performance of Personal Computer Operating Systems. *Division of Applied Sciences*, Harvard University.

[16] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and Performance in the Denali Isolation Kernel. *University of Washington*.

[17] Jeffrey P. Casazza, Michael Greenfield, and Kan Shi. Redefining server performance characterization for virtualization benchmarking. *Intel Technology Journal*, 10(03), 2006.

[18] W. A. Bhat, S. M. K. Quadri. Benchmarking Criteria for File System Benchmark. International Journal of Engineering Science and Technology (IJEST), Page; 665 - 670, India. Jan, 2011.

[19] Keith Adams, Ole Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization. *VMware Inc*.

[20] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauery, Ian Pratt, Andrew Warfield. Xen and the Art of Virtualization. *University of Cambridge Computer Laboratory*, 15 JJ Thomson Avenue, Cambridge, UK.

[21] Andreas Kassler, Andreas Schorr. Generic QOS Aware Media Stream Transcoding and Adaptation. *Department of Distributed Systems, University of Ulm*, Germany.

[22] Eric Cronin, Burton Filstrup, Anthony Kurc. A Distributed Multiplayer Game Server System. *Electrical Engineering and Computer Science Department*, University of Michigan, Ann Arbor, MI 48109-2122, USA. May 4, 2001

[23] M. Vilayannur, Robert B. Ross, R. Thakur, P. H. Carns, S. Anand, Ma. Kandemir, On the Performance of the POSIX I/O Interface to PVFS, Department of Computer Science and Engineering, *Pennsylvania State University*, University Park, USA.

[24] Search Engine sites; http://en.wikipedia.org/wiki/Wikipedia , http://www.google.com

[25] Krishna Bharat, Andrei Broder. Mirror, mirror on the Web: a study of host pairs with replicated content. *Compaq Systems Research Center*, 130 Lytton Avenue, Palo Alto, CA 94301, USA

[26] Andrea Bastoni, Daniel P. Bovet, Marco Cesati, and Paolo Palana. Discovering hypervisor overheads using micro and macrobenchmarks. *System Programming Research Group, University of Rome*, Roma, Italy.

[27] Wasim Ahmad Bhat, S. M. K. Quadri. Benchmarking Criteria for File System Benchmarks. Department of Computer Science, University of Kashmir, India.

[28] Fernando Laudares Camargos, Gabriel Girard, Benoit des Ligneris. A comparative study of Virtualization of Linux servers, July 23rd-26th, 2008, Ottawa, Canada.

[29] VMware Inc. Understanding Full-Virtualization, Para-virtualization and Hardware-Assist. White paper. November 11, 2007.

[30] Larry Mc Voy, Carl Statelin, Lmbench: Portable tools for Performance Analysis, *Silicon Graphics Inc and HP Laboratories*, USA.

[31] Staelin, C., lmbench an extensible micro-benchmark suite, Software: Practice and Experience. *HP Laboratories*, HPL 2004-213, November, 2004.

[32] Charu Chaubal. The Architecture of VMware ESXi. *VMware Inc*. White paper, 2007.

[33] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM:the Linux Virtual Machine Monitor. Proc. of the Linux Symposium, Ottawa, Ontario, Canada: Springer-Verlag, Page 225-230, 2007.

[34] VMware Inc. VMware Infrastructure Architecture Overview. White paper. November 11, 2006.

[35] Stephen Alan Herrod. Systems research and development at VMware. SIGOPS Oper. Syst. Rev., December 2010.

[36] Jianhua Che, Congcong Shi, Yong Yu, Weimin Lin. A Synthetical Performance Evaluation of OpenVZ, Xen and KVM. *2010 IEEE Asia-Pacific Services Computing Conference*, Page 578-594, 2010.

[37] Homepage of Xen documentation. http://www.linux-kvm.org/page/Documents/

[38] Aameek Singh, Madhukar Korupolu, Dushmanta Mohapatra. Server-Storage Virtualization: Integration and Load Balancing in Data Centers. *IBM Almaden Research Center*, USA.

[39] Y.C. Tay. Data Generation for Application-Specific Benchmarking, *National University of Singapore*. Page:1470-1473, Singapore, 2011.

[40] Joseph E. S. Virtualization for Disaster Recovery: areas of focus and consideration. IBM Global Service, USA. Page 2-7, March 2008.

[41] S. Bouckaert, J. V.n Gerwen, I. Moerman, S. Phillips, J. Wilander. Benchmarking computers and computer networks. White Paper.

[42] G. Somani and S. Chaudhary. Application performance isolation in virtualization. *In Cloud Computing, 2009. CLOUD '09. IEEE International Conference* on, pages 41 - 48, sept. 2009.

[43] J. Liedtke, N. Islam, T. Jaeger, Y. Park. Irreproducible Benchmarks Might Be Sometimes Helpful. Thomas J. Watson Research Center, *IBM*, Hawthorne, USA.

[44] Keith Adams, Ole Agesen. A Comparison of Software and Hardware Techniques for x86. *VMware Inc*, Palo Alto, CA, USA.

[45] Sergey Bratus, Michael E. Locasto, Ashwin Ramaswamy, and Sean W. Smith. Traps, Events, Emulation, and Enforcement: Managing the Yin and Yang of Virtualization-based Security. *Dartmouth College*, USA

[46] Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal and Kang G. Shin. Performance Evaluation of Virtualization Technologies for Server Consolidation. *University of Michigan and HP Laboratories*, Palo, Alto, 2007

[47] Lu Liu, Osama Masfary, Jianxin Li. Evaluation of Server Virtualization Technologies for Green IT. *Proceedings of The 6th IEEE International Symposium on Service Oriented System Engineering (SOSE 2011)*, Page 79-84, 2011.

[48] Robert P. Goldberg. Architectural Principles for Virtual Computer Systems. *Deputy for Command and Management Systems*, HQ AFSC, USA. February 1973.

[49] R. J. CREASY. The Origin of the VM/370 Time-sharing System. *IBM Journal. RES. DEVELOP. VOL. 25 NO. 5*, Page 483-490, SEPTEMBER 1981

[50] Diomidis S. and Georgios G.. Beautiful Architecture - Leading Thinkers Reveal the Hidden Beauty in Software Design, USA, 2009

[51] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. T*he University of Washington*, USA

[52] Adeshiyan T., Attanasio C. R., Farr E. M., Harper R. E., Pelleg, D., Schulz, C., Spainhower, L. F. Using virtualization for high availability and disas-

ter recovery. *IBM Journal of Research and Development*. Volume: 53, Page 8:1 - 8:11. 2009.

[53] Ryan Baclit, Chivas Sicam, Peter Membrey, and John Newbigin. Foundations of CentOS Linux, Enterprise Linux On the Cheap. *A book for the expert voice of linux*. November 19, 2009.

[54] Marshall K. M., George V. The Design and Implementation of the FreeBSD Operating System, Fourth printing, June 2008.

[55] Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. Gprof: A call graph execution profiler. *In Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, pages 120-126, USA, 1982.

[56] L. M. Berc, S. A. Leung, J. M. Anderson, J. Dean, S. Ghemawat, M. R. Henzinger, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl. Continuous profiling: where have all the cycles gone? *ACM Transactions on Computer Systems*, page: 357-390, 1997.

[57] V. Anand, H. Franke, H. Linder, S. Nagar, P. Narayanan, R. Ravindra. Benchmarks that Model Enterprise Workloads - using macrobenchmarks to measure and improve Linux scalability for real-world applications. *Proceedings of the Linux Symposium*, Page: 457 - 469, Ontario, Canada , 2003.

Appendix A

Lmbench Output

```
make[1]: Entering directory '/root/lmbench-3.0-a9/results'

                LMBENCH 3.0 SUMMARY
                ___________________

                (Alpha software, do not distribute)

Basic system parameters
_____________________________________________________________________________
Host                 OS Description                Mhz  tlb  cache  mem   scal
                                                       pages line   par   load
                                                             bytes
_________ _____________ ___________________________ ____ _____ _____ ______ ____
dvm1       Linux 2.6.32-          x86_64-linux-gnu 2281    48    64 4.8100    1
Processor, Processes - times in microseconds - smaller is better
_____________________________________________________________________________
Host                 OS  Mhz null null      open slct sig  sig  fork exec sh
                             call  I/O stat clos TCP  inst hndl proc proc proc
_________ _____________ ____ ____ ____ ____ ____ ____ ____ ____ ____ ____ ____
dvm1       Linux 2.6.32- 2281 0.08 0.13 0.90 1.52 2.91 0.17 0.91 194. 512. 1181
Basic integer operations - times in nanoseconds - smaller is better
_____________________________________________________________________
Host                 OS  intgr intgr  intgr  intgr  intgr
                           bit   add    mul    div    mod
_________ _____________ ______ ______ ______ ______ ______
dvm1       Linux 2.6.32- 1.0500 0.9900 0.3200   23.3   13.1
Basic uint64 operations - times in nanoseconds - smaller is better
_____________________________________________________________________
Host                 OS int64  int64  int64  int64  int64
                           bit    add    mul    div    mod
_________ _____________ ______ ______ ______ ______ ______
dvm1       Linux 2.6.32- 0.440         0.1700   33.8   35.1
Basic float operations - times in nanoseconds - smaller is better
_____________________________________________________________________
Host                 OS  float  float  float  float
                           add    mul    div    bogo
```

```
dvm1        Linux 2.6.32-  1.7500 1.7500 8.2100 5.7200
Basic double operations - times in nanoseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS  double double double double
                         add    mul    div    bogo
--------- ------------- ------ ------ ------ ------
dvm1        Linux 2.6.32- 3.8600 3.7300   22.9   15.8
Context switching - times in microseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS  2p/0K 2p/16K 2p/64K 8p/16K 8p/64K 16p/16K 16p/64K
                         ctxsw  ctxsw  ctxsw  ctxsw  ctxsw   ctxsw   ctxsw
--------- ------------- ------ ------ ------ ------ ------ ------- -------
dvm1        Linux 2.6.32- 1.4700 1.7100 3.1200 2.3200 6.6700 2.58000 8.26000
*Local* Communication latencies in microseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS 2p/0K  Pipe AF    UDP  RPC/   TCP  RPC/ TCP
                        ctxsw       UNIX      UDP         TCP conn
--------- ------------- ----- ----- ---- ----- ----- ----- ----- ----
dvm1        Linux 2.6.32- 1.910 4.124 7.19 7.848  13.8  12.2  22.0  35.
*Remote* Communication latencies in microseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS  UDP  RPC/   TCP  RPC/ TCP
                         UDP         TCP  conn
--------- ------------- ----- ----- ----- ----- ----
dvm1        Linux 2.6.32-
File & VM system latencies in microseconds - smaller is better
------------------------------------------------------------------------------
Host                 OS   0K File      10K File     Mmap    Prot    Page   100fd
                        Create Delete Create Delete Latency Fault   Fault  selct
--------- ------------- ------ ------ ------ ------ ------- ------- ------ -----
dvm1  Linux 2.6.32- 7.3324 4.7873   24.8   10.0  6556.0 0.350 1.47780 1.562
*Local* Communication bandwidths in MB/s - bigger is better
------------------------------------------------------------------------------
Host                 OS Pipe AF    TCP  File   Mmap  Bcopy  Bcopy  Mem   Mem
                            UNIX      reread reread (libc) (hand) read  write
--------- ------------- ---- ---- ---- ------ ------ ------ ------ ---- ----
dvm1   inux 2.6.32- 891. 818. 725.  1136.6 1068.5  1735.7  403.6 1567. 902.0
Memory latencies in nanoseconds - smaller is better
    (WARNING - may not be correct, check graphs)
------------------------------------------------------------------------------
Host                 OS   Mhz   L1 $   L2 $    Main mem    Rand mem    Guesses
--------- ------------- ----- ------ ------ ------------ ------------ -------
dvm1        Linux 2.6.32- 2281 1.3150 6.8960         52.7        207.9
make[1]: Leaving directory '/root/lmbench-3.0-a9/results'
```

Appendix B

Bonnie++ Output

```
root@dvm1:~# bonnie++ -u root -d /root/bonnie_tes/t -s 5G
Using uid:0, gid:0.
Writing a byte at a time...done          Writing intelligently...done
Rewriting...done                         Reading a byte at a time...done
Reading intelligently...done
start 'em...done...done...done...done...done...
Create files in sequential order...done.
Stat files in sequential order...done.
Delete files in sequential order...done.
Create files in random order...done.
Stat files in random order...done.
Delete files in random order...done.
Version  1.96       ------Sequential Output------ --Sequential Input- --Random
    -
Concurrency   1     -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks
    --
Machine        Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP  /sec %
    CP
dvm1              5G   753  97  6201   7  2543   0  1086  30  4573   0  53.6
    1
Latency             18667us    14312ms    6978ms     422ms     961ms    2191ms
Version  1.96       ------Sequential Create------ --------Random Create
    --------
dvm1                -Create-- --Read--- -Delete-- -Create-- --Read--- -Delete
    --
              files  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %
                  CP
                 16 13185  16 +++++ +++ 21601  20 21500  23 +++++ +++ 13831
                     14
Latency              6395us     453us     580us     833us     379us     375us
1.96,1.96,dvm1,1,1333281170,5G
    ,,753,97,6201,7,2543,0,1086,30,4573,0,53.6,1,16,,,,,13185,
16,+++++,+++,21601,20,21500,23,+++++,+++,13831,14,18667us,14312ms,6978ms,422ms
    ,961ms,2191ms,6395us,453us,580us,833
```

Appendix C

NetIO Output

```
## Output in Server Machine

TCP server listening.
TCP connection established ...
Receiving from client, packet size  1k ...  6093.67 KByte/s
Sending to client, packet size  1k ...  34.58 MByte/s
Receiving from client, packet size  2k ...  6668.74 KByte/s
Sending to client, packet size  2k ...  35.57 MByte/s
Receiving from client, packet size  4k ...  7505.61 KByte/s
Sending to client, packet size  4k ...  39.27 MByte/s
Receiving from client, packet size  8k ...  7455.51 KByte/s
Sending to client, packet size  8k ...  36.23 MByte/s
Receiving from client, packet size 16k ...  7482.17 KByte/s
Sending to client, packet size 16k ...  39.78 MByte/s
Receiving from client, packet size 32k ...  7824.02 KByte/s
Sending to client, packet size 32k ...  42.57 MByte/s
Done.

## Output in Client Machine

TCP connection established.
Packet size  1k bytes:  6326.67 KByte/s Tx,  981.97 KByte/s Rx.
Packet size  2k bytes:  6632.00 KByte/s Tx,  1058.66 KByte/s Rx.
Packet size  4k bytes:  7046.00 KByte/s Tx,  1144.48 KByte/s Rx.
Packet size  8k bytes:  6960.00 KByte/s Tx,  1221.49 KByte/s Rx.
Packet size 16k bytes:  7778.67 KByte/s Tx,  1278.04 KByte/s Rx.
Packet size 32k bytes:  6173.18 KByte/s Tx,  1443.39 KByte/s Rx.
```

Appendix D

firewall.sh

```
#!/bin/bash
#/root/timer.rc&
IPTABLES=/sbin/iptables

$IPTABLES -F -t filter
$IPTABLES -F -t nat
$IPTABLES -F -t mangle

$IPTABLES --policy INPUT DROP
$IPTABLES --policy FORWARD ACCEPT
$IPTABLES --policy OUTPUT ACCEPT

$IPTABLES -A INPUT  -s 127.0.0.1 -j ACCEPT
$IPTABLES -A OUTPUT -d 127.0.0.1 -j ACCEPT
$IPTABLES -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
$IPTABLES -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

$IPTABLES -A INPUT -i vmbr0 -s 128.39.73.0/24 -j ACCEPT
$IPTABLES -A INPUT -i vmbr0 -s 128.39.74.0/23 -j ACCEPT
$IPTABLES -A INPUT -i vmbr0 -s 128.39.89.0/24 -j ACCEPT
$IPTABLES -A INPUT -i vmbr0 -s 128.39.28.0/22 -j ACCEPT
$IPTABLES -A INPUT -i vmbr0 -s 158.36.144.0/26 -j ACCEPT
$IPTABLES -A INPUT -i vmbr0 -s 158.36.145.128/26 -j ACCEPT
$IPTABLES -A INPUT -i vmbr0 -s 158.36.161.0/24 -j ACCEPT

$IPTABLES -t nat -A POSTROUTING -o vmbr0 -j MASQUERADE

# In TCP throughput test, packet is coming from remote machine to virtual
# machines. But in UDP test, packet is going from local to remote so no
# rule is necessary because default OUTPUT policy is accept.
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4021 -j DNAT --to-
    destination 192.168.1.11:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4022 -j DNAT --to-
    destination 192.168.1.12:18767
```

```
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4023 -j DNAT --to-
    destination 192.168.1.13:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4024 -j DNAT --to-
    destination 192.168.1.14:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4025 -j DNAT --to-
    destination 192.168.1.15:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4026 -j DNAT --to-
    destination 192.168.1.16:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4027 -j DNAT --to-
    destination 192.168.1.17:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4028 -j DNAT --to-
    destination 192.168.1.18:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4029 -j DNAT --to-
    destination 192.168.1.19:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4030 -j DNAT --to-
    destination 192.168.1.20:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4031 -j DNAT --to-
    destination 192.168.1.21:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4032 -j DNAT --to-
    destination 192.168.1.22:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4033 -j DNAT --to-
    destination 192.168.1.23:18767
$IPTABLES -t nat -A PREROUTING -i vmbr0 -p tcp --dport 4034 -j DNAT --to-
    destination 192.168.1.24:18767
```

Appendix E

load.pl

```
#!/usr/bin/perl

$bis = 3292929;
while(TRUE)
{
        for ($i=0; $i<=$bis; $i++)
        {
                $m = 0.000001;
                $n = sin($m);
                $n = $n + 0.00001;
        }
        next;
        $n = $n + 0.01;
}
```